# NEAR EAST UNIVERSITESI

## Faculty of Engineering

## Department of Computer Engineering

# TCP/IP IN NETWORKING
# COM400

**Student:Ayça Sağlam(991160)**

**Supervisor:Assist.Prof.Dr.Firudin Muradov**

**Lefkoşa-2005**

# NEAR EAST UNIVERSITY

Faculty of Engineering

Department of Computer Engineering

TCP/IP IN NETWORKING
COM 400

Student: Ayça Sağlam(991160)

Supervisor: Assist.Prof.Dr.Firudin Muradov

Lefkoşa - 2005

## ACKNOWLEDGEMENTS

First of all, I would like to express my gratitude to academic staff of Engineering Faculty of NEU , especially to Prof. Dr. Fakhraddin Mamedov and Prof. Dr. Senol Bektas.

I wish to thank you my supervisor Assits. Prof. Dr. Firudin Muradov.He is helpfull in my difficulties, he answered my questions and I did them his quidelines.

I would like to express my special thanks to Fevzi Zulaloglu , who was my teacher in high school and supported me whenever I needed his advice.

I am thankfull to my family. Without their endless support and love, I would never be achieved my current position.I wish they live happily always.

## ABSTRACT

Transmisson Control Protocol/Internet Protocol (TCP/IP) is an industry-standard suit of protocols designed for Wide Area Network(WANs).

The standards of TCP/IP, first developed to allow exchange of information between computers in the US government, defence and university research communities.

With the increasing interest in the use of TCP/IP for general commercial applications, there is a need to know what management and technical difficulties will be encountered.

This project is about the practical problems of installing, configuring and maintaining information system based on the TCP/IP set of standards, from initial installation to on-going maintenance.

To be successful and to retain over a long time, the system requires frequent revision on initial assumptions, system designers must take account not only of technical, but also of social and organizational problems they will encounter. While succesful system grow and develop, it used for purposes that the initial design probably did not predict. Once convenient and reliable operation is achieved, the users abandon and then lose the older, less convinient alternatives. The new system becomes part of the corporate infrastructure and day-to-day life, its value increases and any change in performance and availability can dramatically affect prosperity and well-being.

TABLE OF CONTENTS

## INTRODUCTION

An increasing number of people are using the Internet and, many for the first time, are using the tools and utilities that at one time were only available on a limited number of computer systems (and only for really intense users!). One sign of this growth in use has been the significant number of TCP/IP and Internet books, articles, courses, and even TV shows that have become available in the last several years; there are so many such books that publishers are reluctant to authorize more because bookstores have reached their limit of shelf space! This memo provides a broad overview of the Internet and TCP/IP, with an emphasis on history, terms, and concepts. It is meant as a brief guide and starting point, referring to many other sources for more detailed information.

This project introduces TCP/IP with any level of computer skills or computer background knowledge. Writing this project is to explain in a simple way some concepts that may be considered difficult. This project leads a TCP/IP beginner to an intermediate understanding of TCP/IP. In this project each topic is covered to sufficient depth but not to an extreme.

This project present the relevant material, and I've included what I have found to be the most important concepts. This project filled with several simple examples, diagrams, and screen captures in an effort to make the TCP/IP protocol. Graphics in this project for understanding easily.

This project is neither operating system-specific nor software-specific. Concepts are presented so that the reader can gain an understanding of the topic without being tied to a particular platform. In this project TCP/IP are programmer guides to TCP/IP.

Anyone studying for a TCP/IP exam will find this project useful for fine-tuning any concepts that they do not thoroughly understand.

Someone who may be interested in a particular topic within TCP/IP can pick up the project and get a quick, thorough understanding. Many executives and IS decision-makers need to be conversant with TCP/IP so that they can talk with their staff and other professionals. This is the project provide that understanding.

## CHAPTER ONE:INTRODUCTION TO NETWORK

### 1.1 Network Hardware

It is now time to turn our attention to the technical issues involved in network design (the work stuff). There is no generally accepted taxonomy into which all computer networks fit, but two dimensions stand out as important: transmission technology and scale. We will now examine each of these in turn.

Broadly speaking, there are two types of transmission technology that are in widespread use. They are as follows:

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

Broadcast networks have a single communication channel that is shared by all the machines on the network. Short messages, called packets in certain contexts, sent by any machine are received by all the others. An address field within the packet specifies the intended recipient. Upon receiving a packet, a machine checks the address field. If the packet is intended for the receiving machine, that machine processes the packet; if the packet is intended for some other machine, it is just ignored.

As an analogy, consider someone standing at the end of a corridor with many rooms off it and shouting "Watson, come here. I want you." Although the packet may actually be received (heard) by many people, only Watson responds. The others just ignore it. Another analogy is an airport announcement asking all flight 644 passengers to report to gate 12 for immediate boarding.

Broadcast systems generally also allow the possibility of addressing a packet to all destinations by using a special code in the address field. When a packet with this code is transmitted, it is received and processed by every machine on the network. This mode of operation is called broadcasting. Some broadcast systems also support transmission to a subset of the machines, something known as multicasting. One possible scheme is to reserve one bit to indicate multicasting. The remaining n - 1 address bits can hold a group number. Each machine can "subscribe" to any or all of the groups. When a packet is sent to a certain group, it is delivered to all machines subscribing to that group.

In contrast, point-to-point networks consist of many connections between individual pairs of machines. To go from the source to the destination, a packet on this type of network may have to first visit one or more intermediate machines. Often multiple routes, of different lengths, are possible, so finding good ones is important in point-to-point networks. As a general rule (although there are many exceptions), smaller, geographically localized networks tend to use broadcasting, whereas larger networks usually are point-to-point. Point-to-point transmission with one sender and one receiver is sometimes called unicasting.

An alternative criterion for classifying networks is their scale. In Fig. 1-6 we classify multiple processor systems by their physical size. At the top are the personal area networks, networks that are meant for one person. For example, a wireless network connecting a computer with its mouse, keyboard, and printer is a personal area network. Also, a PDA that controls the user's hearing aid or pacemaker fits in this category. Beyond the personal area networks come longer-range networks. These can be divided into local, metropolitan, and wide area networks. Finally, the connection of two or more networks is called an internetwork. The worldwide Internet is a well-known example of an internetwork. Distance is important as a classification metric because different techniques are used at different scales. Below we give a brief introduction to network hardware.

***Figure 1-1. Classification of interconnected processors by scale.***

| Interprocessor distance | Processors located in same | Example |
|---|---|---|
| 1 m | Square meter | Personal area network |
| 10 m | Room | Local area network |
| 100 m | Building | |
| 1 km | Campus | |
| 10 km | City | Metropolitan area network |
| 100 km | Country | Wide area network |
| 1000 km | Continent | |
| 10,000 km | Planet | The Internet |

### 1.1.1 Local Area Networks

Local area networks, generally called LANs, are privately-owned networks within a single building or campus of up to a few kilometers in size. They are widely used to connect personal computers and workstations in company offices and factories to share resources

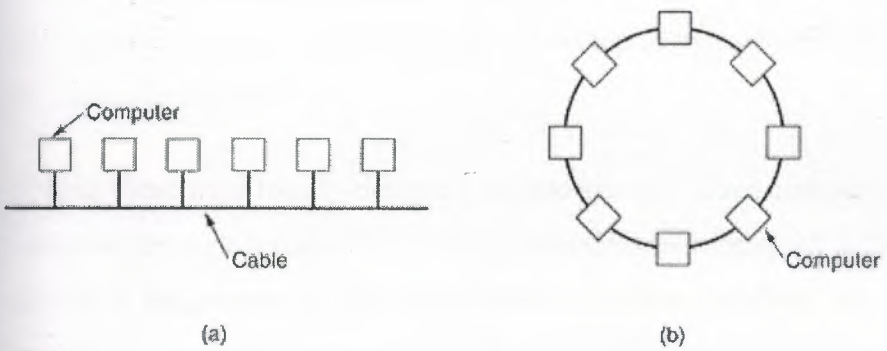
(e.g., printers) and exchange information. LANs are distinguished from other kinds of networks by three characteristics: (1) their size, (2) their transmission technology, and (3) their topology.

LANs are restricted in size, which means that the worst-case transmission time is bounded and known in advance. Knowing this bound makes it possible to use certain kinds of designs that would not otherwise be possible. It also simplifies network management.

LANs may use a transmission technology consisting of a cable to which all the machines are attached, like the telephone company party lines once used in rural areas. Traditional LANs run at speeds of 10 Mbps to 100 Mbps, have low delay (microseconds or nanoseconds), and make very few errors. Newer LANs operate at up to 10 Gbps.

Various topologies are possible for broadcast LANs. Figure 1-2 shows two of them. In a bus (i.e., a linear cable) network, at any instant at most one machine is the master and is allowed to transmit. All other machines are required to refrain from sending. An arbitration mechanism is needed to resolve conflicts when two or more machines want to transmit simultaneously. The arbitration mechanism may be centralized or distributed. IEEE 802.3, popularly called Ethernet, for example, is a bus-based broadcast network with decentralized control, usually operating at 10 Mbps to 10 Gbps. Computers on an Ethernet can transmit whenever they want to; if two or more packets collide, each computer just waits a random time and tries again later.

***Figure 1-2. Two broadcast networks. (a) Bus. (b) Ring.***




A second type of broadcast system is the ring. In a ring, each bit propagates around on its own, not waiting for the rest of the packet to which it belongs. Typically, each bit circumnavigates the entire ring in the time it takes to transmit a few bits, often before the

complete packet has even been transmitted. As with all other broadcast systems, some rule is needed for arbitrating simultaneous accesses to the ring. Various methods, such as having the machines take turns, are in use. IEEE 802.5 (the IBM token ring), is a ring-based LAN operating at 4 and 16 Mbps. FDDI is another example of a ring network.

Broadcast networks can be further divided into static and dynamic, depending on how the channel is allocated. A typical static allocation would be to divide time into discrete intervals and use a round-robin algorithm, allowing each machine to broadcast only when its time slot comes up. Static allocation wastes channel capacity when a machine has nothing to say during its allocated slot, so most systems attempt to allocate the channel dynamically (i.e., on demand).

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

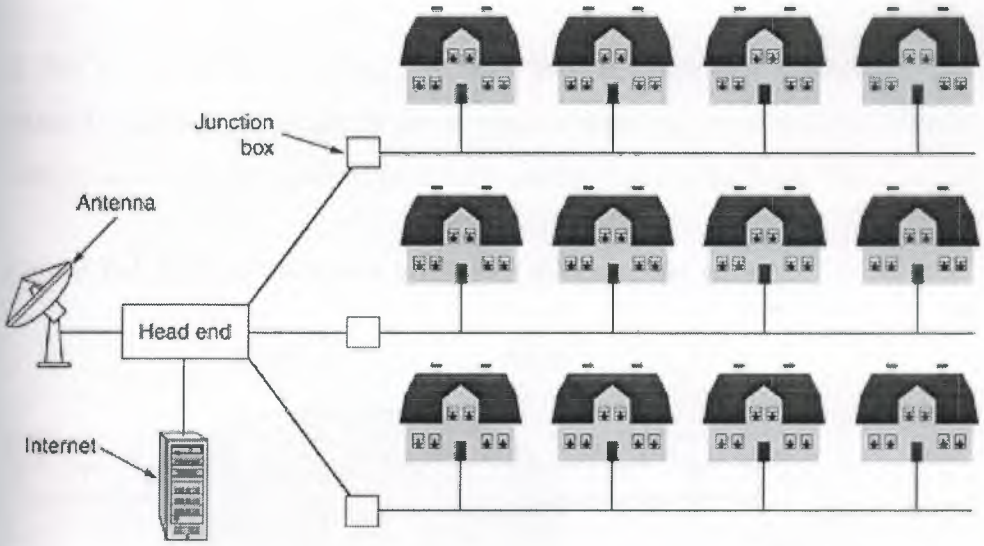
### 1.1.2 Metropolitan Area Networks

A metropolitan area network, or MAN, covers a city. The best-known example of a MAN is the cable television network available in many cities. This system grew from earlier community antenna systems used in areas with poor over-the-air television reception. In these early systems, a large antenna was placed on top of a nearby hill and signal was then piped to the subscribers' houses.

At first, these were locally-designed, ad hoc systems. Then companies began jumping into the business, getting contracts from city governments to wire up an entire city. The next step was television programming and even entire channels designed for cable only. Often these channels were highly specialized, such as all news, all sports, all cooking, all gardening, and so on. But from their inception until the late 1990s, they were intended for television reception only.

Starting when the Internet attracted a mass audience, the cable TV network operators began to realize that with some changes to the system, they could provide two-way Internet service in unused parts of the spectrum. At that point, the cable TV system began to morph from a way to distribute television to a metropolitan area network. To a first approximation, a MAN might look something like the system shown in Fig. 1-3. In this figure we see both television signals and Internet being fed into the centralized head end for subsequent distribution to people's homes.

***Figure 1-3. A metropolitan area network based on cable TV.***




Cable television is not the only MAN. Recent developments in high-speed wireless Internet access resulted in another MAN, which has been standardized as IEEE 802.16.

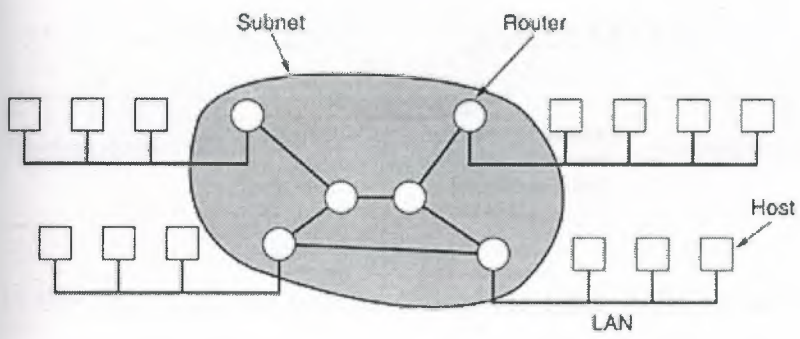
### 1.1.3 Wide Area Networks

A wide area network, or WAN, spans a large geographical area, often a country or continent. It contains a collection of machines intended for running user (i.e., application) programs. We will follow traditional usage and call these machines hosts. The hosts are connected by a communication subnet, or just subnet for short. The hosts are owned by the customers (e.g., people's personal computers), whereas the communication subnet is typically owned and operated by a telephone company or Internet service provider. The job of the subnet is to carry messages from host to host, just as the telephone system carries words from speaker to listener. Separation of the pure communication aspects of the network (the subnet) from the application aspects (the hosts), greatly simplifies the complete network design.

In most wide area networks, the subnet consists of two distinct components: transmission lines and switching elements. Transmission lines move bits between machines. They can be made of copper wire, optical fiber, or even radio links. Switching elements are specialized computers that connect three or more transmission lines. When data arrive on an incoming line, the switching element must choose an outgoing line on which to forward them. These switching computers have been called by various names in the past; the name router is now most commonly used. Unfortunately, some people pronounce it "rooter" and others have it rhyme with "doubter." Determining the correct pronunciation will be left as an exercise for the reader. (Note: the perceived correct answer may depend on where you live.)

In this model, shown in Fig. 1-4, each host is frequently connected to a LAN on which a router is present, although in some cases a host can be connected directly to a router. The collection of communication lines and routers (but not the hosts form the subnet).

***Figure 1-4. Relation between hosts on LANs and the subnet.***



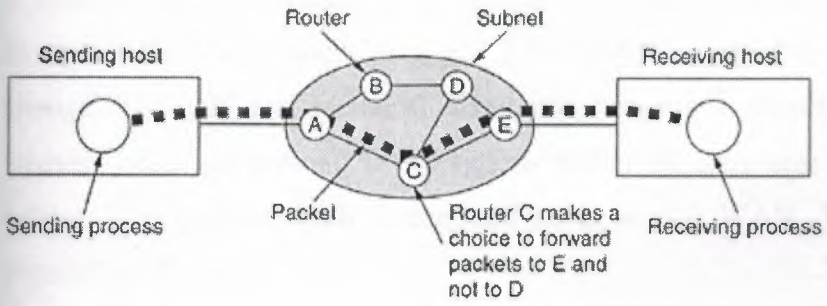

A short comment about the term "subnet" is in order here. Originally, its only meaning was the collection of routers and communication lines that moved packets from the source host to the destination host. However, some years later, it also acquired a second meaning in conjunction with network addressing . Unfortunately, no widely-used alternative exists for its initial meaning, so with some hesitation we will use it in both senses. From the context, it will always be clear which is meant.

In most WANs, the network contains numerous transmission lines, each one connecting a pair of routers. If two routers that do not share a transmission line wish to communicate, they must do this indirectly, via other routers. When a packet is sent from one router to another via one or more intermediate routers, the packet is received at each intermediate router in its entirety, stored there until the required output line is free, and then forwarded. A subnet organized

according to this principle is called a store-and-forward or packet-switched subnet. Nearly all wide area networks (except those using satellites) have store-and-forward subnets. When the packets are small and all the same size, they are often called cells.

The principle of a packet-switched WAN is so important that it is worth devoting a few more words to it. Generally, when a process on some host has a message to be sent to a process on some other host, the sending host first cuts the message into packets, each one bearing its number in the sequence. These packets are then injected into the network one at a time in quick succession. The packets are transported individually over the network and deposited at the receiving host, where they are reassembled into the original message and delivered to the receiving process. A stream of packets resulting from some initial message is illustrated in Fig. 1-5.

***Figure 1-5. A stream of packets from sender to receiver.***




In this figure, all the packets follow the route ACE, rather than ABDE or ACDE. In some networks all packets from a given message must follow the same route; in others each packet is routed separately. Of course, if ACE is the best route, all packets may be sent along it, even if each packet is individually routed.

Routing decisions are made locally. When a packet arrives at router A,itis up to A to decide if this packet should be sent on the line to B or the line to C. How A makes that decision is called the routing algorithm. Many of them exist.

Not all WANs are packet switched. A second possibility for a WAN is a satellite system. Each router has an antenna through which it can send and receive. All routers can hear the output from the satellite, and in some cases they can also hear the upward transmissions of their fellow routers to the satellite as well. Sometimes the routers are connected to a substantial point-to-point subnet, with only some of them having a satellite antenna. Satellite

networks are inherently broadcast and are most useful when the broadcast property is important.

### 1.1.4 Internetworks

Many networks exist in the world, often with different hardware and software. People connected to one network often want to communicate with people attached to a different one. The fulfillment of this desire requires that different, and frequently incompatible networks, be connected, sometimes by means of machines called gateways to make the connection and provide the necessary translation, both in terms of hardware and software. A collection of interconnected networks is called an internetwork or internet. These terms will be used in a generic sense, in contrast to the worldwide Internet (which is one specific internet), which we will always capitalize.

A common form of internet is a collection of LANs connected by a WAN. In fact, if we were to replace the label "subnet" in Fig. 1-5 by "WAN," nothing else in the figure would have to change. The only real technical distinction between a subnet and a WAN in this case is whether hosts are present. If the system within the gray area contains only routers, it is a subnet; if it contains both routers and hosts, it is a WAN. The real differences relate to ownership and use.

Subnets, networks, and internetworks are often confused. Subnet makes the most sense in the context of a wide area network, where it refers to the collection of routers and communication lines owned by the network operator. As an analogy, the telephone system consists of telephone switching offices connected to one another by high-speed lines, and to houses and businesses by low-speed lines. These lines and equipment, owned and managed by the telephone company, form the subnet of the telephone system. The telephones themselves (the hosts in this analogy) are not part of the subnet. The combination of a subnet and its hosts forms a network. In the case of a LAN, the cable and the hosts form the network. There really is no subnet.

An internetwork is formed when distinct networks are interconnected. In our view, connecting a LAN and a WAN or connecting two LANs forms an internetwork, but there is little agreement in the industry over terminology in this area. One rule of thumb is that if different organizations paid to construct different parts of the network and each maintains its part, we

have an internetwork rather than a single network. Also, if the underlying technology is different in different parts (e.g., broadcast versus point-to-point), we probably have two networks. Internet, contrasting nicely with it. Next we will introduce Ethernet, the dominant local area network

## 1.2 Network Software

The first computer networks were designed with the hardware as the main concern and the software as an afterthought. This strategy no longer works. Network software is now highly structured. In the following sections we examine the software structuring technique in some detail. The method described here forms the keystone of the entire book and will occur repeatedly later on.

### 1.2.1 Protocol Hierarchies

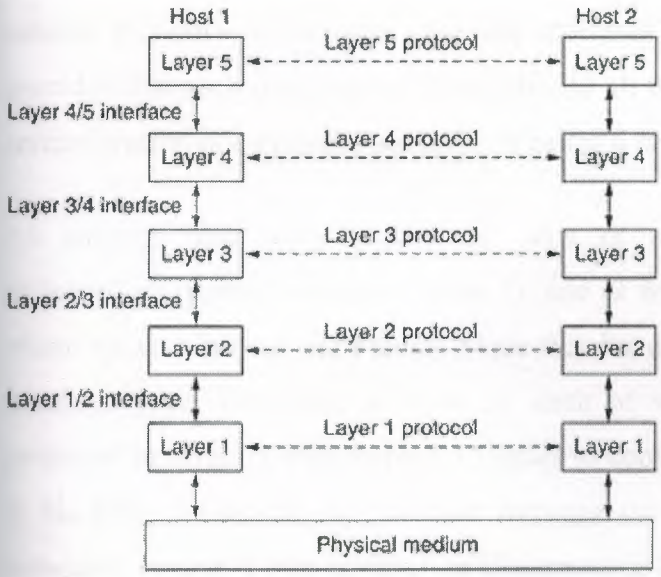
To reduce their design complexity, most networks are organized as a stack of layers or levels, each one built upon the one below it. The number of layers, the name of each layer, the contents of each layer, and the function of each layer differ from network to network. The purpose of each layer is to offer certain services to the higher layers, shielding those layers from the details of how the offered services are actually implemented. In a sense, each layer is a kind of virtual machine, offering certain services to the layer above it.

This concept is actually a familiar one and used throughout computer science, where it is variously known as information hiding, abstract data types, data encapsulation, and object-oriented programming. The fundamental idea is that a particular piece of software (or hardware) provides a service to its users but keeps the details of its internal state and algorithms hidden from them.

Layer n on one machine carries on a conversation with layer n on another machine. The rules and conventions used in this conversation are collectively known as the layer n protocol. Basically, a protocol is an agreement between the communicating parties on how communication is to proceed. As an analogy, when a woman is introduced to a man, she may choose to stick out her hand. He, in turn, may decide either to shake it or kiss it, depending, for example, on whether she is an American lawyer at a business meeting or a European princess at a formal ball. Violating the protocol will make communication more difficult, if not completely impossible.

A five-layer network is illustrated in Fig. 1-6. The entities comprising the corresponding layers on different machines are called peers. The peers may be processes, hardware devices, or even human beings. In other words, it is the peers that communicate by using the protocol.

***Figure 1-6. Layers, protocols, and interfaces.***



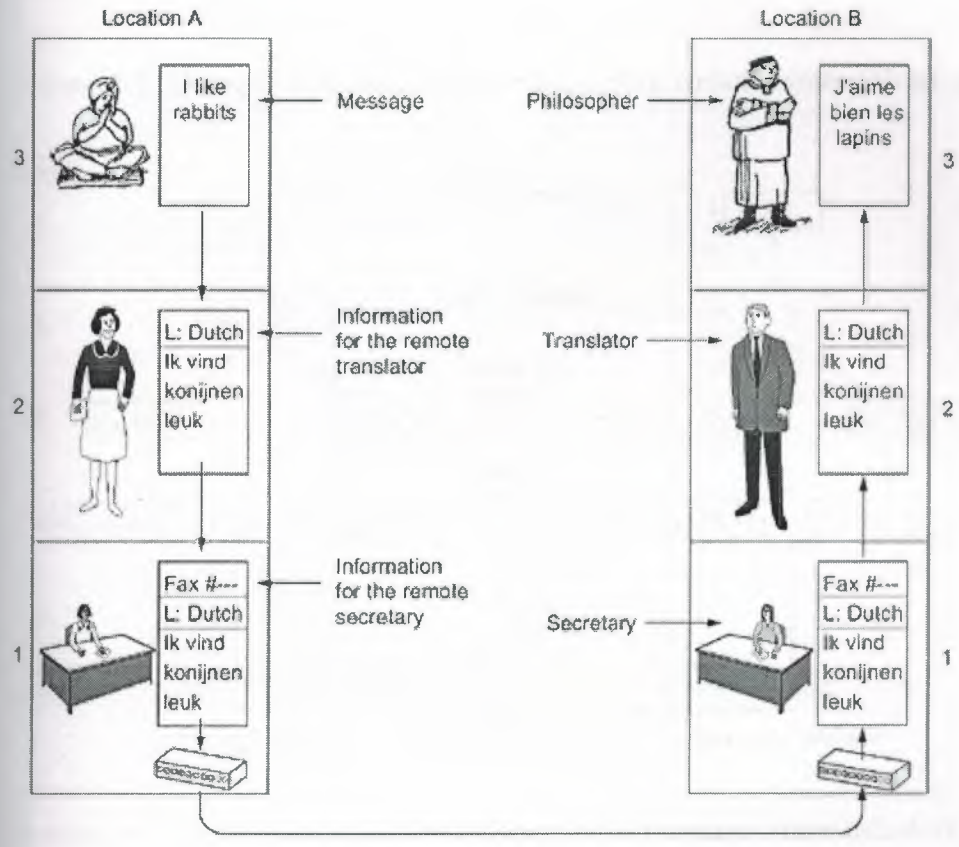

In reality, no data are directly transferred from layer n on one machine to layer n on another machine. Instead, each layer passes data and control information to the layer immediately below it, until the lowest layer is reached. Below layer 1 is the physical medium through which actual communication occurs. In Fig. 1-6, virtual communication is shown by dotted lines and physical communication by solid lines.

Between each pair of adjacent layers is an interface. The interface defines which primitive operations and services the lower layer makes available to the upper one. When network designers decide how many layers to include in a network and what each one should do, one of the most important considerations is defining clean interfaces between the layers. Doing so, in turn, requires that each layer perform a specific collection of well-understood functions. In addition to minimizing the amount of information that must be passed between layers, clear-cut interfaces also make it simpler to replace the implementation of one layer with a completely different implementation (e.g., all the telephone lines are replaced by satellite channels) because all that is required of the new implementation is that it offer exactly the same set of services to its upstairs neighbor as the old implementation did. In fact, it is common that different hosts use different implementations.

A set of layers and protocols is called a network architecture. The specification of an architecture must contain enough information to allow an implementer to write the program or build the hardware for each layer so that it will correctly obey the appropriate protocol. Neither the details of the implementation nor the specification of the interfaces is part of the architecture because these are hidden away inside the machines and not visible from the outside. It is not even necessary that the interfaces on all machines in a network be the same, provided that each machine can correctly use all the protocols. A list of protocols used by a certain system, one protocol per layer, is called a protocol stack.

An analogy may help explain the idea of multilayer communication. Imagine two philosophers (peer processes in layer 3), one of whom speaks Urdu and English and one of whom speaks Chinese and French. Since they have no common language, they each engage a translator (peer processes at layer 2), each of whom in turn contacts a secretary (peer processes in layer 1). Philosopher 1 wishes to convey his affection for oryctolagus cuniculus to his peer. To do so, he passes a message (in English) across the 2/3 interface to his translator, saying "I like rabbits," as illustrated in Fig. 1-7. The translators have agreed on a neutral language known to both of them, Dutch, so the message is converted to "Ik vind konijnen leuk." The choice of language is the layer 2 protocol and is up to the layer 2 peer processes.

**Figure 1-7. The philosopher-translator-secretary architecture.**

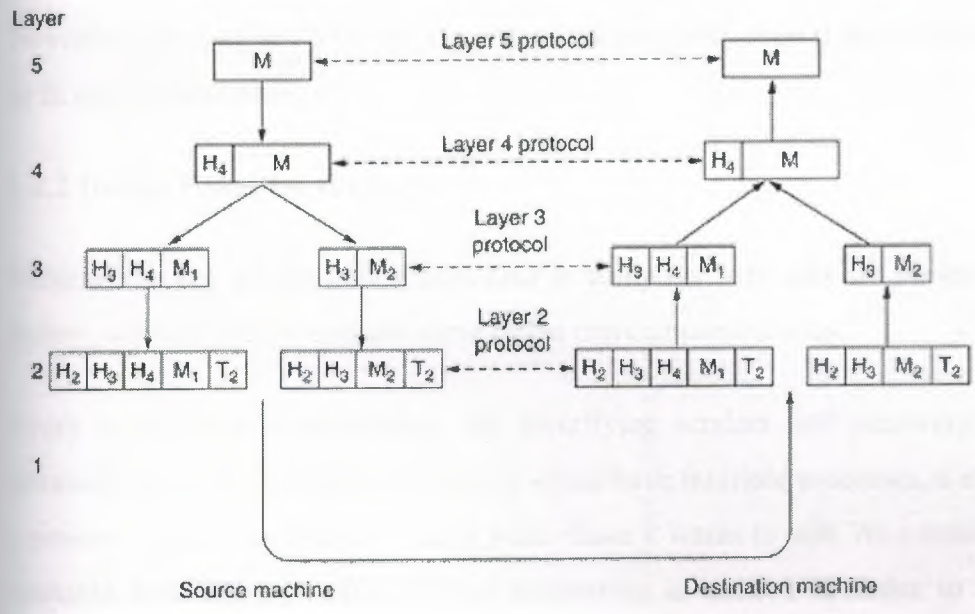



The translator then gives the message to a secretary for transmission, by, for example, fax (the layer 1 protocol). When the message arrives, it is translated into French and passed across the 2/3 interface to philosopher 2. Note that each protocol is completely independent of the other ones as long as the interfaces are not changed. The translators can switch from Dutch to say, Finnish, at will, provided that they both agree, and neither changes his interface with either layer 1 or layer 3. Similarly, the secretaries can switch from fax to e-mail or telephone without disturbing (or even informing) the other layers. Each process may add some information intended only for its peer. This information is not passed upward to the layer above.

Now consider a more technical example: how to provide communication to the top layer of the five-layer network in Fig. 1-8. A message, M, is produced by an application process running in layer 5 and given to layer 4 for transmission. Layer 4 puts a header in front of the message to identify the message and passes the result to layer 3. The header includes control information, such as sequence numbers, to allow layer 4 on the destination machine to deliver

messages in the right order if the lower layers do not maintain sequence. In some layers, headers can also contain sizes, times, and other control fields.

***Figure 1-8. Example information flow supporting virtual communication in layer 5.***




In many networks, there is no limit to the size of messages transmitted in the layer 4 protocol, but there is nearly always a limit imposed by the layer 3 protocol. Consequently, layer 3 must break up the incoming messages into smaller units, packets, prepending a layer 3 header to each packet. In this example, M is split into two parts, $M_1$ and $M_2$.

Layer 3 decides which of the outgoing lines to use and passes the packets to layer 2. Layer 2 adds not only a header to each piece, but also a trailer, and gives the resulting unit to layer 1 for physical transmission. At the receiving machine the message moves upward, from layer to layer, with headers being stripped off as it progresses. None of the headers for layers below n are passed up to layer n.

The important thing to understand about Fig. 1-8 is the relation between the virtual and actual communication and the difference between protocols and interfaces. The peer processes in layer 4, for example, conceptually think of their communication as being "horizontal," using the layer 4 protocol. Each one is likely to have a procedure called something like SendToOtherSide and GetFromOtherSide, even though these procedures actually communicate with lower layers across the 3/4 interface, not with the other side.

The peer process abstraction is crucial to all network design. Using it, the unmanageable task of designing the complete network can be broken into several smaller, manageable design problems, namely, the design of the individual layers.

Lower layers of a protocol hierarchy are frequently implemented in hardware or firmware. Nevertheless, complex protocol algorithms are involved, even if they are embedded (in whole or in part) in hardware.

### 1.2.2 Design Issues for the Layers

Some of the key design issues that occur in computer networks are present in several layers. Below, we will briefly mention some of the more important ones.

Every layer needs a mechanism for identifying senders and receivers. Since a network normally has many computers, some of which have multiple processes, a means is needed for a process on one machine to specify with whom it wants to talk. As a consequence of having multiple destinations, some form of addressing is needed in order to specify a specific destination.

Another set of design decisions concerns the rules for data transfer. In some systems, data only travel in one direction; in others, data can go both ways. The protocol must also determine how many logical channels the connection corresponds to and what their priorities are. Many networks provide at least two logical channels per connection, one for normal data and one for urgent data.

Error control is an important issue because physical communication circuits are not perfect. Many error-detecting and error-correcting codes are known, but both ends of the connection must agree on which one is being used. In addition, the receiver must have some way of telling the sender which messages have been correctly received and which have not.

Not all communication channels preserve the order of messages sent on them. To deal with a possible loss of sequencing, the protocol must make explicit provision for the receiver to allow the pieces to be reassembled properly. An obvious solution is to number the pieces, but this solution still leaves open the question of what should be done with pieces that arrive out of order.

An issue that occurs at every level is how to keep a fast sender from swamping a slow receiver with data. Various solutions have been proposed and will be discussed later. Some of them involve some kind of feedback from the receiver to the sender, either directly or indirectly, about the receiver's current situation. Others limit the sender to an agreed-on transmission rate. This subject is called flow control.

Another problem that must be solved at several levels is the inability of all processes to accept arbitrarily long messages. This property leads to mechanisms for disassembling, transmitting, and then reassembling messages. A related issue is the problem of what to do when processes insist on transmitting data in units that are so small that sending each one separately is inefficient. Here the solution is to gather several small messages heading toward a common destination into a single large message and dismember the large message at the other side.

When it is inconvenient or expensive to set up a separate connection for each pair of communicating processes, the underlying layer may decide to use the same connection for multiple, unrelated conversations. As long as this multiplexing and demultiplexing is done transparently, it can be used by any layer. Multiplexing is needed in the physical layer, for example, where all the traffic for all connections has to be sent physical layer, for example, where all the traffic for all connections has to be sent over at most a few physical circuits.

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

### 1.2.3 Connection-Oriented and Connectionless Services

Layers can offer two different types of service to the layers above them: connection-oriented and connectionless. In this section we will look at these two types and examine the differences between them.

Connection-oriented service is modeled after the telephone system. To talk to someone, you pick up the phone, dial the number, talk, and then hang up. Similarly, to use a connection-oriented network service, the service user first establishes a connection, uses the connection,

and then releases the connection. The essential aspect of a connection is that it acts like a tube: the sender pushes objects (bits) in at one end, and the receiver takes them out at the other end. In most cases the order is preserved so that the bits arrive in the order they were sent.

In some cases when a connection is established, the sender, receiver, and subnet conduct a negotiation about parameters to be used, such as maximum message size, quality of service required, and other issues. Typically, one side makes a proposal and the other side can accept it, reject it, or make a counterproposal.

In contrast, connectionless service is modeled after the postal system. Each message (letter) carries the full destination address, and each one is routed through the system independent of all the others. Normally, when two messages are sent to the same destination, the first one sent will be the first one to arrive. However, it is possible that the first one sent can be delayed so that the second one arrives first.

Each service can be characterized by a quality of service. Some services are reliable in the sense that they never lose data. Usually, a reliable service is implemented by having the receiver acknowledge the receipt of each message so the sender is sure that it arrived. The acknowledgement process introduces overhead and delays, which are often worth it but are sometimes undesirable.

A typical situation in which a reliable connection-oriented service is appropriate is file transfer. The owner of the file wants to be sure that all the bits arrive correctly and in the same order they were sent. Very few file transfer customers would prefer a service that occasionally scrambles or loses a few bits, even if it is much faster.

Reliable connection-oriented service has two minor variations: message sequences and byte streams. In the former variant, the message boundaries are preserved. When two 1024-byte messages are sent, they arrive as two distinct 1024-byte messages, never as one 2048-byte message. In the latter, the connection is simply a stream of bytes, with no message boundaries. When 2048 bytes arrive at the receiver, there is no way to tell if they were sent as one 2048-byte message, two 1024-byte messages, or 2048 1-byte messages. If the pages of a book are sent over a network to a phototypesetter as separate messages, it might be important to preserve the message boundaries. On the other hand, when a user logs into a remote server,

a byte stream from the user's computer to the server is all that is needed. Message boundaries are not relevant.

As mentioned above, for some applications, the transit delays introduced by acknowledgements are unacceptable. One such application is digitized voice traffic. It is preferable for telephone users to hear a bit of noise on the line from time to time than to experience a delay waiting for acknowledgements. Similarly, when transmitting a video conference, having a few pixels wrong is no problem, but having the image jerk along as the flow stops to correct errors is irritating.

Not all applications require connections. For example, as electronic mail becomes more common, electronic junk is becoming more common too. The electronic junk-mail sender probably does not want to go to the trouble of setting up and later tearing down a connection just to send one item. Nor is 100 percent reliable delivery essential, especially if it costs more. All that is needed is a way to send a single message that has a high probability of arrival, but no guarantee. Unreliable (meaning not acknowledged) connectionless service is often called datagram service, in analogy with telegram service, which also does not return an acknowledgement to the sender.

In other situations, the convenience of not having to establish a connection to send one short message is desired, but reliability is essential. The acknowledged datagram service can be provided for these applications. It is like sending a registered letter and requesting a return receipt. When the receipt comes back, the sender is absolutely sure that the letter was delivered to the intended party and not lost along the way.

Still another service is the request-reply service. In this service the sender transmits a single datagram containing a request; the reply contains the answer. For example, a query to the local library asking where Uighur is spoken falls into this category. Request-reply is commonly used to implement communication in the client-server model: the client issues a request and the server responds to it. Figure 1-9 summarizes the types of services discussed above.

*Figure 1-9. Six different types of service.*

| | Service | Example |
|---|---|---|
| Connection-oriented | Reliable message stream | Sequence of pages |
| | Reliable byte stream | Remote login |
| | Unreliable connection | Digitized voice |
| Connectionless | Unreliable datagram | Electronic junk mail |
| | Acknowledged datagram | Registered mail |
| | Request-reply | Database query |

The concept of using unreliable communication may be confusing at first. After all, why would anyone actually prefer unreliable communication to reliable communication? First of all, reliable communication (in our sense, that is, acknowledged) may not be available. For example, Ethernet does not provide reliable communication. Packets can occasionally be damaged in transit. It is up to higher protocol levels to deal with this problem. Second, the delays inherent in providing a reliable service may be unacceptable, especially in real-time applications such as multimedia. For these reasons, both reliable and unreliable communication coexist.

### 1.2.4 Service Primitives

A service is formally specified by a set of primitives (operations) available to a user process to access the service. These primitives tell the service to perform some action or report on an action taken by a peer entity. If the protocol stack is located in the operating system, as it often is, the primitives are normally system calls. These calls cause a trap to kernel mode, which then turns control of the machine over to the operating system to send the necessary packets.

The set of primitives available depends on the nature of the service being provided. The primitives for connection-oriented service are different from those of connectionless service. As a minimal example of the service primitives that might be provided to implement a reliable byte stream in a client-server environment, consider the primitives listed in Fig. 1-10.

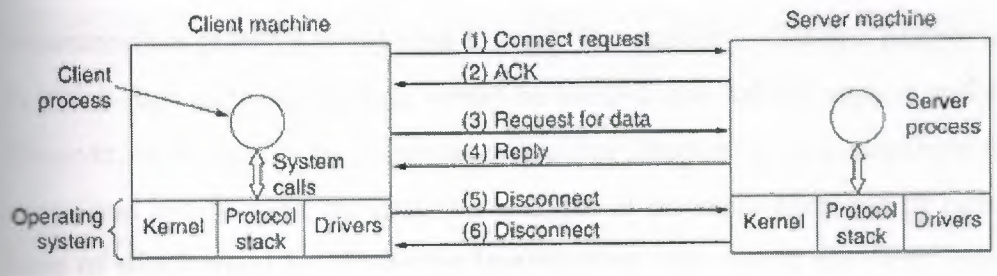
*Figure 1-10. Five service primitives for implementing a simple connection-oriented service.*

| Primitive | Meaning |
|---|---|
| LISTEN | Block waiting for an incoming connection |
| CONNECT | Establish a connection with a waiting peer |
| RECEIVE | Block waiting for an incoming message |
| SEND | Send a message to the peer |
| DISCONNECT | Terminate a connection |

These primitives might be used as follows. First, the server executes LISTEN to indicate that it is prepared to accept incoming connections. A common way to implement LISTEN is to make it a blocking system call. After executing the primitive, the server process is blocked until a request for connection appears.

Next, the client process executes CONNECT to establish a connection with the server. The CONNECT call needs to specify who to connect to, so it might have a parameter giving the server's address. The operating system then typically sends a packet to the peer asking it to connect, as shown by (1) in Fig. 1-11. The client process is suspended until there is a response. When the packet arrives at the server, it is processed by the operating system there. When the system sees that the packet is requesting a connection, it checks to see if there is a listener. If so, it does two things: unblocks the listener and sends back an acknowledgement (2). The arrival of this acknowledgement then releases the client. At this point the client and server are both running and they have a connection established. It is important to note that the acknowledgement (2) is generated by the protocol code itself, not in response to a user-level primitive. If a connection request arrives and there is no listener, the result is undefined. In some systems the packet may be queued for a short time in anticipation of a LISTEN.

*Figure 1-11. Packets sent in a simple client-server interaction on a connection-oriented network.*



The obvious analogy between this protocol and real life is a customer (client) calling a company's customer service manager. The service manager starts out by being near the telephone in case it rings. Then the client places the call. When the manager picks up the phone, the connection is established.

The next step is for the server to execute RECEIVE to prepare to accept the first request. Normally, the server does this immediately upon being released from the LISTEN, before the acknowledgement can get back to the client. The RECEIVE call blocks the server.

Then the client executes SEND to transmit its request (3) followed by the execution of RECEIVE to get the reply.

The arrival of the request packet at the server machine unblocks the server process so it can process the request. After it has done the work, it uses SEND to return the answer to the client (4). The arrival of this packet unblocks the client, which can now inspect the answer. If the client has additional requests, it can make them now. If it is done, it can use DISCONNECT to terminate the connection. Usually, an initial DISCONNECT is a blocking call, suspending the client and sending a packet to the server saying that the connection is no longer needed (5). When the server gets the packet, it also issues a DISCONNECT of its own, acknowledging the client and releasing the connection. When the server's packet (6) gets back to the client machine, the client process is released and the connection is broken. In a nutshell, this is how connection-oriented communication works.

Of course, life is not so simple. Many things can go wrong here. The timing can be wrong (e.g., the CONNECT is done before the LISTEN), packets can get lost, and much more. We will look at these issues in great detail later, but for the moment, Fig. 1-11 briefly summarizes how client-server communication might work over a connection-oriented network.

Given that six packets are required to complete this protocol, one might wonder why a connectionless protocol is not used instead. The answer is that in a perfect world it could be, in which case only two packets would be needed: one for the request and one for the reply. However, in the face of large messages in either direction (e.g., a megabyte file), transmission errors, and lost packets, the situation changes. If the reply consisted of hundreds of packets, some of which could be lost during transmission, how would the client know if some pieces were missing? How would the client know whether the last packet actually received was

really the last packet sent? Suppose that the client wanted a second file. How could it tell packet 1 from the second file from a lost packet 1 from the first file that suddenly found its way to the client? In short, in the real world, a simple request-reply protocol over an unreliable network is often inadequate. Later we will study a variety of protocols in detail that overcome these and other problems. For the moment, suffice it to say that having a reliable, ordered byte stream between processes is sometimes very convenient.

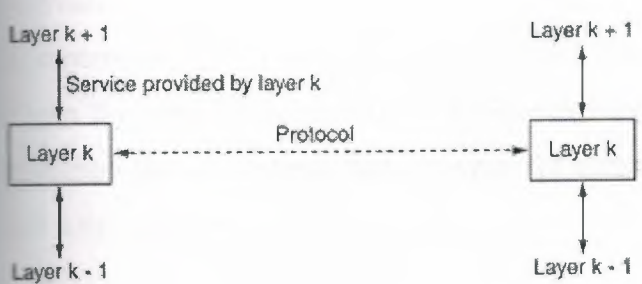
### 1.2.5 The Relationship of Services to Protocols

Services and protocols are distinct concepts, although they are frequently confused. This distinction is so important, however, that we emphasize it again here. A service is a set of primitives (operations) that a layer provides to the layer above it. The service defines what operations the layer is prepared to perform on behalf of its users, but it says nothing at all about how these operations are implemented. A service relates to an interface between two layers, with the lower layer being the service provider and the upper layer being the service user.

A protocol, in contrast, is a set of rules governing the format and meaning of the packets, or messages that are exchanged by the peer entities within a layer. Entities use protocols to implement their service definitions. They are free to change their protocols at will, provided they do not change the service visible to their users. In this way, the service and the protocol are completely decoupled.

In other words, services relate to the interfaces between layers, as illustrated in Fig. 1-12. In contrast, protocols relate to the packets sent between peer entities on different machines. It is important not to confuse the two concepts.

***Figure 1-12. The relationship between a service and a protocol.***



An analogy with programming languages is worth making. A service is like an abstract data type or an object in an object-oriented language. It defines operations that can be performed on an object but does not specify how these operations are implemented. A protocol relates to the implementation of the service and as such is not visible to the user of the service.

Many older protocols did not distinguish the service from the protocol. In effect, a typical layer might have had a service primitive SEND PACKET with the user providing a pointer to a fully assembled packet. This arrangement meant that all changes to the protocol were immediately visible to the users. Most network designers now regard such a design as a serious blunder.

## CHAPTER TWO:OSI AND TCP/IP REFERENCE MODELS

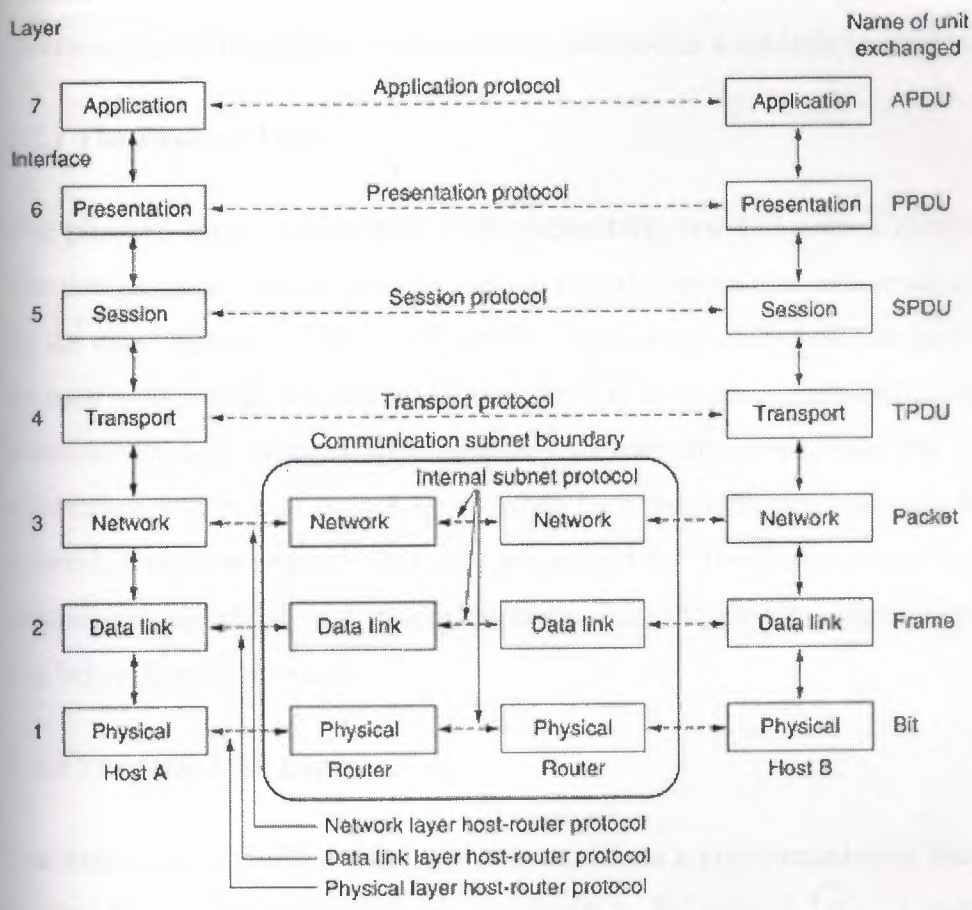
### 2.1 Reference Models

Now that we have discussed layered networks in the abstract, it is time to look at some examples. We will discuss two important network architectures, the OSI reference model and the TCP/IP reference model. Although the protocols associated with the OSI model are rarely used any more, the model itself is actually quite general and still valid, and the features discussed at each layer are still very important. The TCP/IP model has the opposite properties: the model itself is not of much use but the protocols are widely used. For this reason we will look at both of them in detail. Also, sometimes you can learn more from failures than from successes.

### 2.1 The OSI Reference Model

The OSI model (minus the physical medium) is shown in Fig. 2-1. This model is based on a proposal developed by the International Standards Organization (ISO) as a first step toward international standardization of the protocols used in the various layers (Day and Zimmermann, 1983). It was revised in 1995 (Day, 1995). The model is called the ISO OSI (Open Systems Interconnection) Reference Model because it deals with connecting open systems—that is, systems that are open for communication with other systems. We will just call it the OSI model for short.

*Figure 2-1. The OSI reference model.*




The OSI model has seven layers. The principles that were applied to arrive at the seven layers can be briefly summarized as follows:

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

Below we will discuss each layer of the model in turn, starting at the bottom layer. Note that the OSI model itself is not a network architecture because it does not specify the exact

services and protocols to be used in each layer. It just tells what each layer should do. However, ISO has also produced standards for all the layers, although these are not part of the reference model itself. Each one has been published as a separate international standard.

### *2.2.1 The Physical Layer*

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

### *2.2.2 The Data Link Layer*

The main task of the data link layer is to transform a raw transmission facility into a line that appears free of undetected transmission errors to the network layer. It accomplishes this task by having the sender break up the input data into data frames (typically a few hundred or a few thousand bytes) and transmit the frames sequentially. If the service is reliable, the receiver confirms correct receipt of each frame by sending back an acknowledgement frame.

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

Broadcast networks have an additional issue in the data link layer: how to control access to the shared channel. A special sublayer of the data link layer, the medium access control sublayer, deals with this problem.

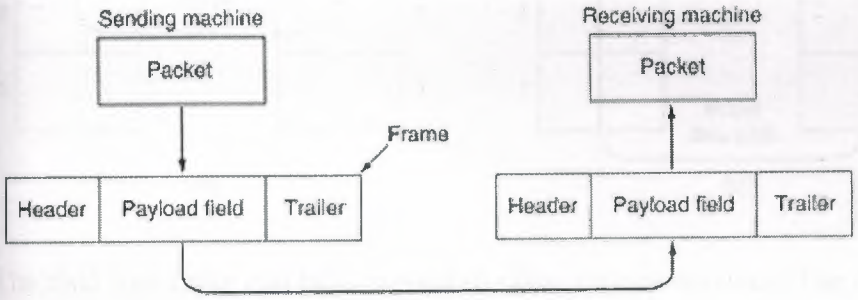
### Data Link Layer Design Issues

The data link layer has a number of specific functions it can carry out. These functions include

1. Providing a well-defined service interface to the network layer.
2. Dealing with transmission errors.
3. Regulating the flow of data so that slow receivers are not swamped by fast senders.

To accomplish these goals, the data link layer takes the packets it gets from the network layer and encapsulates them into frames for transmission. Each frame contains a frame header, a payload field for holding the packet, and a frame trailer, as illustrated in Fig. 2-2. Frame management forms the heart of what the data link layer does. In the following sections we will examine all the above-mentioned issues in detail.

***Figure 2-2. Relationship between packets and frames.***



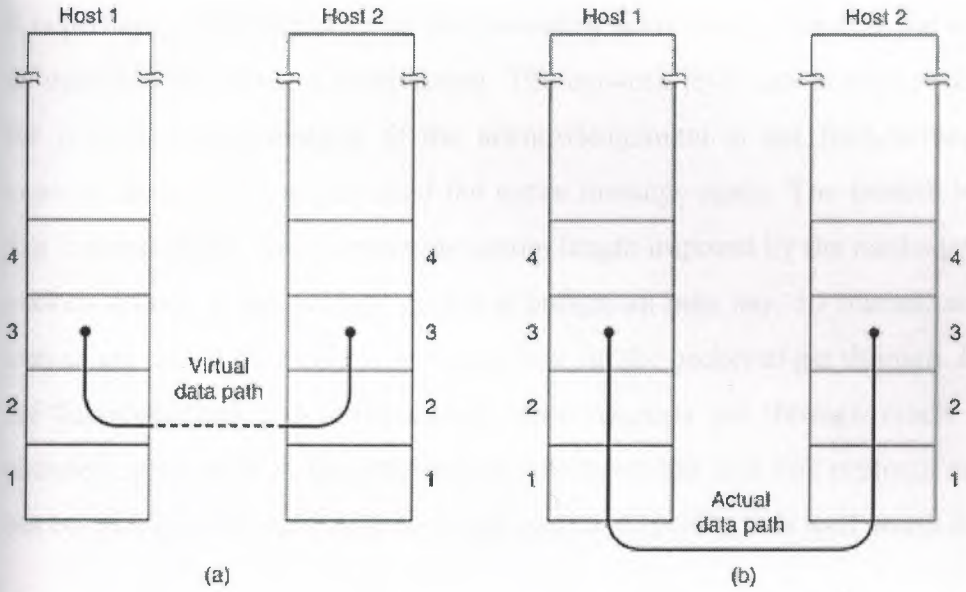

Although this chapter is explicitly about the data link layer and the data link protocols, many of the principles we will study here, such as error control and flow control, are found in transport and other protocols as well. In fact, in many networks, these functions are found only in the upper layers and not in the data link layer. However, no matter where they are found, the principles are pretty much the same, so it does not really matter where we study them. In the data link layer they often show up in their simplest and purest forms, making this a good place to examine them in detail.

### Services Provided to the Network Layer

The function of the data link layer is to provide services to the network layer. The principal service is transferring data from the network layer on the source machine to the network layer on the destination machine. On the source machine is an entity, call it a process, in the

network layer that hands some bits to the data link layer for transmission to the destination. The job of the data link layer is to transmit the bits to the destination machine so they can be handed over to the network layer there, as shown in Fig. 2-3a). The actual transmission follows the path of Fig. 2-3(b), but it is easier to think in terms of two data link layer processes communicating using a data link protocol. For this reason, we will implicitly use the model of Fig. 2-3(a) throughout this chapter.

***Figure 2-3. (a) Virtual communication. (b) Actual communication.***




The data link layer can be designed to offer various services. The actual services offered can vary from system to system. Three reasonable possibilities that are commonly provided are

1. Unacknowledged connectionless service.
2. Acknowledged connectionless service.
3. Acknowledged connection-oriented service.

Let us consider each of these in turn.

Unacknowledged connectionless service consists of having the source machine send independent frames to the destination machine without having the destination machine acknowledge them. No logical connection is established beforehand or released afterward. If a frame is lost due to noise on the line, no attempt is made to detect the loss or recover from it in the data link layer. This class of service is appropriate when the error rate is very low so that recovery is left to higher layers. It is also appropriate for real-time traffic, such as voice,

in which late data are worse than bad data. Most LANs use unacknowledged connectionless service in the data link layer.

The next step up in terms of reliability is acknowledged connectionless service. When this service is offered, there are still no logical connections used, but each frame sent is individually acknowledged. In this way, the sender knows whether a frame has arrived correctly. If it has not arrived within a specified time interval, it can be sent again. This service is useful over unreliable channels, such as wireless systems.

It is perhaps worth emphasizing that providing acknowledgements in the data link layer is just an optimization, never a requirement. The network layer can always send a packet and wait for it to be acknowledged. If the acknowledgement is not forthcoming before the timer expires, the sender can just send the entire message again. The trouble with this strategy is that frames usually have a strict maximum length imposed by the hardware and network layer packets do not. If the average packet is broken up into, say, 10 frames, and 20 percent of all frames are lost, it may take a very long time for the packet to get through. If individual frames are acknowledged and retransmitted, entire packets get through much faster. On reliable channels, such as fiber, the overhead of a heavyweight data link protocol may be unnecessary, but on wireless channels, with their inherent unreliability, it is well worth the cost.

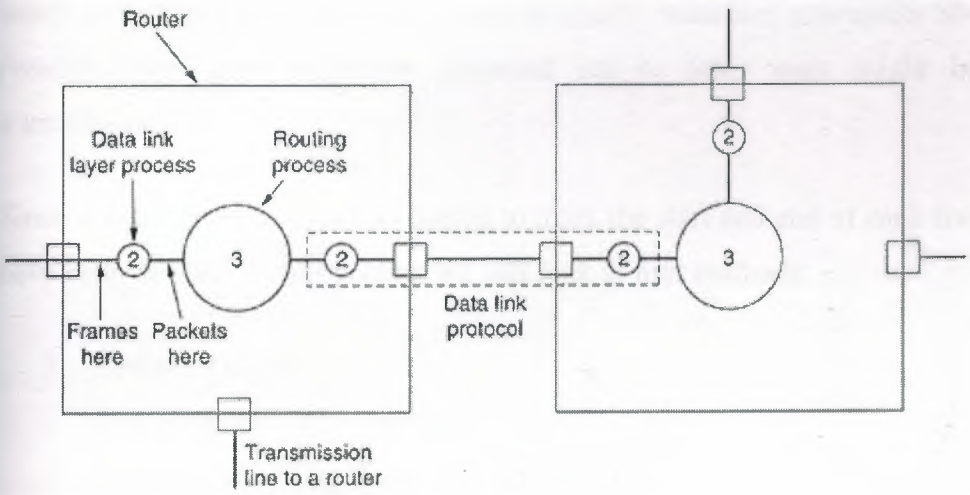
Getting back to our services, the most sophisticated service the data link layer can provide to the network layer is connection-oriented service. With this service, the source and destination machines establish a connection before any data are transferred. Each frame sent over the connection is numbered, and the data link layer guarantees that each frame sent is indeed received. Furthermore, it guarantees that each frame is received exactly once and that all frames are received in the right order. With connectionless service, in contrast, it is conceivable that a lost acknowledgement causes a packet to be sent several times and thus received several times. Connection-oriented service, in contrast, provides the network layer processes with the equivalent of a reliable bit stream.

When connection-oriented service is used, transfers go through three distinct phases. In the first phase, the connection is established by having both sides initialize variables and counters needed to keep track of which frames have been received and which ones have not. In the second phase, one or more frames are actually transmitted. In the third and final phase, the

connection is released, freeing up the variables, buffers, and other resources used to maintain the connection.

Consider a typical example: a WAN subnet consisting of routers connected by point-to-point leased telephone lines. When a frame arrives at a router, the hardware checks it for errors (using techniques we will study late in this chapter), then passes the frame to the data link layer software (which might be embedded in a chip on the network interface board). The data link layer software checks to see if this is the frame expected, and if so, gives the packet contained in the payload field to the routing software. The routing software then chooses the appropriate outgoing line and passes the packet back down to the data link layer software, which then transmits it. The flow over two routers is shown in Fig. 2-4.

***Figure 2-4. Placement of the data link protocol.***




The routing code frequently wants the job done right, that is, with reliable, sequenced connections on each of the point-to-point lines. It does not want to be bothered too often with packets that got lost on the way. It is up to the data link protocol, shown in the dotted rectangle, to make unreliable communication lines look perfect or, at least, fairly good. As an aside, although we have shown multiple copies of the data link layer software in each router, in fact, one copy handles all the lines, with different tables and data structures for each one.

**Framing**

To provide service to the network layer, the data link layer must use the service provided to it by the physical layer. What the physical layer does is accept a raw bit stream and attempt to

deliver it to the destination. This bit stream is not guaranteed to be error free. The number of bits received may be less than, equal to, or more than the number of bits transmitted, and they may have different values. It is up to the data link layer to detect and, if necessary, correct errors.

The usual approach is for the data link layer to break the bit stream up into discrete frames and compute the checksum for each frame. (Checksum algorithms will be discussed later in this chapter.) When a frame arrives at the destination, the checksum is recomputed. If the newly-computed checksum is different from the one contained in the frame, the data link layer knows that an error has occurred and takes steps to deal with it (e.g., discarding the bad frame and possibly also sending back an error report).

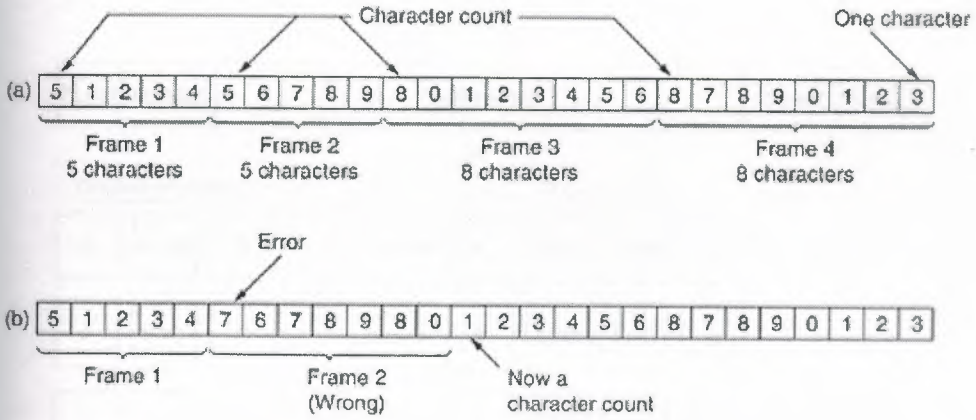
Breaking the bit stream up into frames is more difficult than it at first appears. One way to achieve this framing is to insert time gaps between frames, much like the spaces between words in ordinary text. However, networks rarely make any guarantees about timing, so it is possible these gaps might be squeezed out or other gaps might be inserted during transmission.

Since it is too risky to count on timing to mark the start and end of each frame, other methods have been devised. In this section we will look at four methods:

1. Character count.
2. Flag bytes with byte stuffing.
3. Starting and ending flags, with bit stuffing.
4. Physical layer coding violations.

The first framing method uses a field in the header to specify the number of characters in the frame. When the data link layer at the destination sees the character count, it knows how many characters follow and hence where the end of the frame is. This technique is shown in Fig. 2-5(a) for four frames of sizes 5, 5, 8, and 8 characters, respectively.

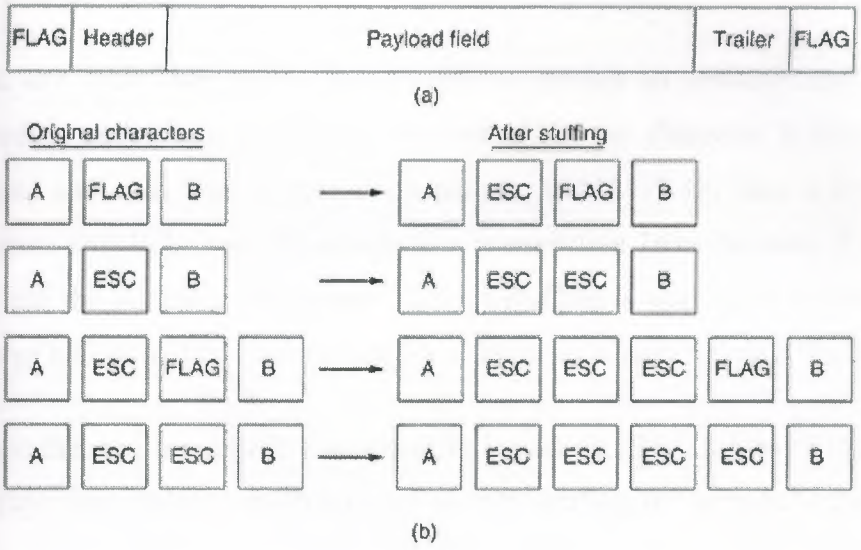
*Figure 2-5. A character stream. (a) Without errors. (b) With one error.*




The trouble with this algorithm is that the count can be garbled by a transmission error. For example, if the character count of 5 in the second frame of Fig. 2-5(b) becomes a 7, the destination will get out of synchronization and will be unable to locate the start of the next frame. Even if the checksum is incorrect so the destination knows that the frame is bad, it still has no way of telling where the next frame starts. Sending a frame back to the source asking for a retransmission does not help either, since the destination does not know how many characters to skip over to get to the start of the retransmission. For this reason, the character count method is rarely used anymore.

The second framing method gets around the problem of resynchronization after an error by having each frame start and end with special bytes. In the past, the starting and ending bytes were different, but in recent years most protocols have used the same byte, called a flag byte, as both the starting and ending delimiter, as shown in Fig. 2-6(a) as FLAG. In this way, if the receiver ever loses synchronization, it can just search for the flag byte to find the end of the current frame. Two consecutive flag bytes indicate the end of one frame and start of the next one.

*Figure 2-6. (a) A frame delimited by flag bytes. (b) Four examples of byte sequences before and after byte stuffing.*




A serious problem occurs with this method when binary data, such as object programs or floating-point numbers, are being transmitted. It may easily happen that the flag byte's bit pattern occurs in the data. This situation will usually interfere with the framing. One way to solve this problem is to have the sender's data link layer insert a special escape byte (ESC) just before each "accidental" flag byte in the data. The data link layer on the receiving end removes the escape byte before the data are given to the network layer. This technique is called byte stuffing or character stuffing. Thus, a framing flag byte can be distinguished from one in the data by the absence or presence of an escape byte before it.

Of course, the next question is: What happens if an escape byte occurs in the middle of the data? The answer is that it, too, is stuffed with an escape byte. Thus, any single escape byte is part of an escape sequence, whereas a doubled one indicates that a single escape occurred naturally in the data. Some examples are shown in Fig. 2-6(b). In all cases, the byte sequence delivered after destuffing is exactly the same as the original byte sequence.

The byte-stuffing scheme depicted in Fig. 2-6 is a slight simplification of the one used in the PPP protocol that most home computers use to communicate with their Internet service provider. We will discuss PPP later in this chapter.

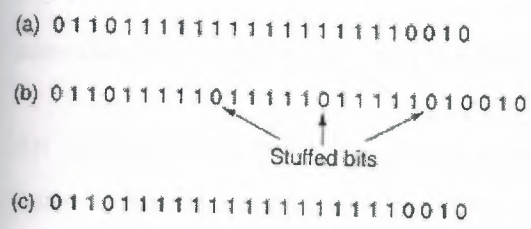
A major disadvantage of using this framing method is that it is closely tied to the use of 8-bit characters. Not all character codes use 8-bit characters. For example. UNICODE uses 16-bit

characters, As networks developed, the disadvantages of embedding the character code length in the framing mechanism became more and more obvious, so a new technique had to be developed to allow arbitrary sized characters.

The new technique allows data frames to contain an arbitrary number of bits and allows character codes with an arbitrary number of bits per character. It works like this. Each frame begins and ends with a special bit pattern, 01111110 (in fact, a flag byte). Whenever the sender's data link layer encounters five consecutive 1s in the data, it automatically stuffs a 0 bit into the outgoing bit stream. This bit stuffing is analogous to byte stuffing, in which an escape byte is stuffed into the outgoing character stream before a flag byte in the data.

When the receiver sees five consecutive incoming 1 bits, followed by a 0 bit, it automatically destuffs (i.e., deletes) the 0 bit. Just as byte stuffing is completely transparent to the network layer in both computers, so is bit stuffing. If the user data contain the flag pattern, 01111110, this flag is transmitted as 011111010 but stored in the receiver's memory as 01111110. Figure 2-7 gives an example of bit stuffing.

***Figure 2-7. Bit stuffing. (a) The original data. (b) The data as they appear on the line. (c) The data as they are stored in the receiver's memory after destuffing.***




With bit stuffing, the boundary between two frames can be unambiguously recognized by the flag pattern. Thus, if the receiver loses track of where it is, all it has to do is scan the input for flag sequences, since they can only occur at frame boundaries and never within the data.

The last method of framing is only applicable to networks in which the encoding on the physical medium contains some redundancy. For example, some LANs encode 1 bit of data by using 2 physical bits. Normally, a 1 bit is a high-low pair and a 0 bit is a low-high pair. The scheme means that every data bit has a transition in the middle, making it easy for the receiver to locate the bit boundaries. The combinations high-high and low-low are not used for data but are used for delimiting frames in some protocols.

As a final note on framing, many data link protocols use a combination of a character count with one of the other methods for extra safety. When a frame arrives, the count field is used to locate the end of the frame. Only if the appropriate delimiter is present at that position and the checksum is correct is the frame accepted as valid. Otherwise, the input stream is scanned for the next delimiter.

**Error Control**

Having solved the problem of marking the start and end of each frame, we come to the next problem: how to make sure all frames are eventually delivered to the network layer at the destination and in the proper order. Suppose that the sender just kept outputting frames without regard to whether they were arriving properly. This might be fine for unacknowledged connectionless service, but would most certainly not be fine for reliable, connection-oriented service.

The usual way to ensure reliable delivery is to provide the sender with some feedback about what is happening at the other end of the line. Typically, the protocol calls for the receiver to send back special control frames bearing positive or negative acknowledgements about the incoming frames. If the sender receives a positive acknowledgement about a frame, it knows the frame has arrived safely. On the other hand, a negative acknowledgement means that something has gone wrong, and the frame must be transmitted again.

An additional complication comes from the possibility that hardware troubles may cause a frame to vanish completely (e.g., in a noise burst). In this case, the receiver will not react at all, since it has no reason to react. It should be clear that a protocol in which the sender transmits a frame and then waits for an acknowledgement, positive or negative, will hang forever if a frame is ever lost due to, for example, malfunctioning hardware.

This possibility is dealt with by introducing timers into the data link layer. When the sender transmits a frame, it generally also starts a timer. The timer is set to expire after an interval long enough for the frame to reach the destination, be processed there, and have the acknowledgement propagate back to the sender. Normally, the frame will be correctly received and the acknowledgement will get back before the timer runs out, in which case the timer will be canceled.

However, if either the frame or the acknowledgement is lost, the timer will go off, alerting the sender to a potential problem. The obvious solution is to just transmit the frame again. However, when frames may be transmitted multiple times there is a danger that the receiver will accept the same frame two or more times and pass it to the network layer more than once. To prevent this from happening, it is generally necessary to assign sequence numbers to outgoing frames, so that the receiver can distinguish retransmissions from originals.

The whole issue of managing the timers and sequence numbers so as to ensure that each frame is ultimately passed to the network layer at the destination exactly once, no more and no less, is an important part of the data link layer's duties. Later in this chapter, we will look at a series of increasingly sophisticated examples to see how this management is done.

**Flow Control**

Another important design issue that occurs in the data link layer (and higher layers as well) is what to do with a sender that systematically wants to transmit frames faster than the receiver can accept them. This situation can easily occur when the sender is running on a fast (or lightly loaded) computer and the receiver is running on a slow (or heavily loaded) machine. The sender keeps pumping the frames out at a high rate until the receiver is completely swamped. Even if the transmission is error free, at a certain point the receiver will simply be unable to handle the frames as they arrive and will start to lose some. Clearly, something has to be done to prevent this situation.

Two approaches are commonly used. In the first one, feedback-based flow control, the receiver sends back information to the sender giving it permission to send more data or at least telling the sender how the receiver is doing. In the second one, rate-based flow control, the protocol has a built-in mechanism that limits the rate at which senders may transmit data, without using feedback from the receiver. In this chapter we will study feedback-based flow control schemes because rate-based schemes are never used in the data link layer.

Various feedback-based flow control schemes are known, but most of them use the same basic principle. The protocol contains well-defined rules about when a sender may transmit the next frame. These rules often prohibit frames from being sent until the receiver has granted permission, either implicitly or explicitly. For example, when a connection is set up,

the receiver might say: "You may send me n frames now, but after they have been sent, do not send any more until I have told you to continue." We will examine the details shortly.

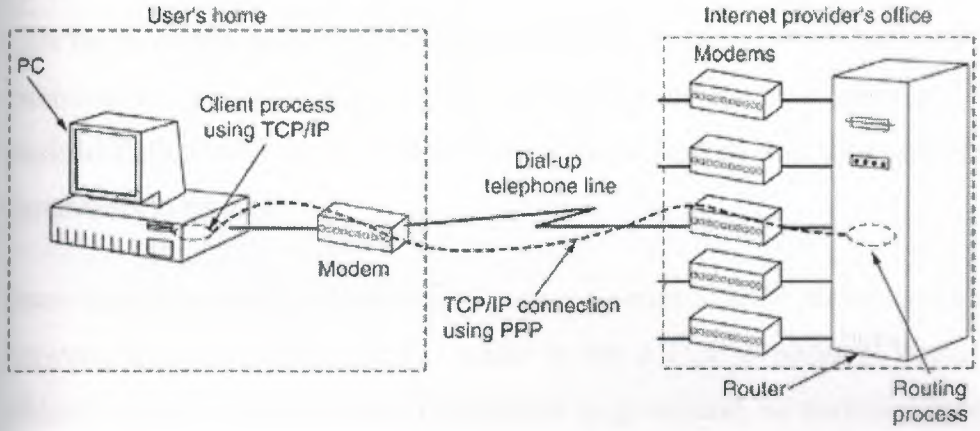
**The Data Link Layer in the Internet**

The Internet consists of individual machines (hosts and routers) and the communication infrastructure that connects them. Within a single building, LANs are widely used for interconnection, but most of the wide area infrastructure is built up from point-to-point leased lines. We will look at LANs; here we will examine the data link protocols used on point-to-point lines in the Internet.

In practice, point-to-point communication is primarily used in two situations. First, thousands of organizations have one or more LANs, each with some number of hosts (personal computers, user workstations, servers, and so on) along with a router (or a bridge, which is functionally similar). Often, the routers are interconnected by a backbone LAN. Typically, all connections to the outside world go through one or two routers that have point-to-point leased lines to distant routers. It is these routers and their leased lines that make up the communication subnets on which the Internet is built.

The second situation in which point-to-point lines play a major role in the Internet is the millions of individuals who have home connections to the Internet using modems and dial-up telephone lines. Usually, what happens is that the user's home PC calls up an Internet service provider's router and then acts like a full-blown Internet host. This method of operation is no different from having a leased line between the PC and the router, except that the connection is terminated when the user ends the session. A home PC calling an Internet service provider is illustrated in Fig. 2-8. The modem is shown external to the computer to emphasize its role, but modern computers have internal modems.

*Figure 2-8. A home personal computer acting as an Internet host.*




For both the router-router leased line connection and the dial-up host-router connection, some point-to-point data link protocol is required on the line for framing, error control, and the other data link layer functions we have studied in this chapter. The one used in the Internet is called PPP. We will now examine it.

### *PPP—The Point-to-Point Protocol*

The Internet needs a point-to-point protocol for a variety of purposes, including router-to-router traffic and home user-to-ISP traffic. This protocol is PPP (Point-to-Point Protocol), which is defined in RFC 1661 and further elaborated on in several other RFCs (e.g., RFCs 1662 and 1663). PPP handles error detection, supports multiple protocols, allows IP addresses to be negotiated at connection time, permits authentication, and has many other features.

PPP provides three features:

1. A framing method that unambiguously delineates the end of one frame and the start of the next one. The frame format also handles error detection.
2. A link control protocol for bringing lines up, testing them, negotiating options, and bringing them down again gracefully when they are no longer needed. This protocol is called LCP (Link Control Protocol). It supports synchronous and asynchronous circuits and byte-oriented and bit-oriented encodings.
3. A way to negotiate network-layer options in a way that is independent of the network layer protocol to be used. The method chosen is to have a different NCP (Network Control Protocol) for each network layer supported.

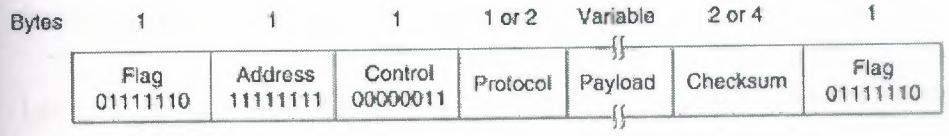
To see how these pieces fit together, let us consider the typical scenario of a home user calling up an Internet service provider to make a home PC a temporary Internet host. The PC first calls the provider's router via a modem. After the router's modem has answered the phone and established a physical connection, the PC sends the router a series of LCP packets in the payload field of one or more PPP frames. These packets and their responses select the PPP parameters to be used.

Once the parameters have been agreed upon, a series of NCP packets are sent to configure the network layer. Typically, the PC wants to run a TCP/IP protocol stack, so it needs an IP address. There are not enough IP addresses to go around, so normally each Internet provider gets a block of them and then dynamically assigns one to each newly attached PC for the duration of its login session. If a provider owns n IP addresses, it can have up to n machines logged in simultaneously, but its total customer base may be many times that. The NCP for IP assigns the IP address.

At this point, the PC is now an Internet host and can send and receive IP packets, just as hardwired hosts can. When the user is finished, NCP tears down the network layer connection and frees up the IP address. Then LCP shuts down the data link layer connection. Finally, the computer tells the modem to hang up the phone, releasing the physical layer connection.

The PPP frame format was chosen to closely resemble the HDLC frame format, since there was no reason to reinvent the wheel. The major difference between PPP and HDLC is that PPP is character oriented rather than bit oriented. In particular, PPP uses byte stuffing on dial-up modem lines, so all frames are an integral number of bytes. It is not possible to send a frame consisting of 30.25 bytes, as it is with HDLC. Not only can PPP frames be sent over dial-up telephone lines, but they can also be sent over SONET or true bit-oriented HDLC lines (e.g., for router-router connections). The PPP frame format is shown in Fig. 2-9.

***Figure 2-9. The PPP full frame format for unnumbered mode operation.***




All PPP frames begin with the standard HDLC flag byte (01111110), which is byte stuffed if it occurs within the payload field. Next comes the Address field, which is always set to the

binary value 11111111 to indicate that all stations are to accept the frame. Using this value avoids the issue of having to assign data link addresses.

The Address field is followed by the Control field, the default value of which is 00000011. This value indicates an unnumbered frame. In other words, PPP does not provide reliable transmission using sequence numbers and acknowledgements as the default. In noisy environments, such as wireless networks, reliable transmission using numbered mode can be used. The exact details are defined in RFC 1663, but in practice it is rarely used.

Since the Address and Control fields are always constant in the default configuration, LCP provides the necessary mechanism for the two parties to negotiate an option to just omit them altogether and save 2 bytes per frame.

The fourth PPP field is the Protocol field. Its job is to tell what kind of packet is in the Payload field. Codes are defined for LCP, NCP, IP, IPX, AppleTalk, and other protocols. Protocols starting with a 0 bit are network layer protocols such as IP, IPX, OSI CLNP, XNS. Those starting with a 1 bit are used to negotiate other protocols. These include LCP and a different NCP for each network layer protocol supported. The default size of the Protocol field is 2 bytes, but it can be negotiated down to 1 byte using LCP.

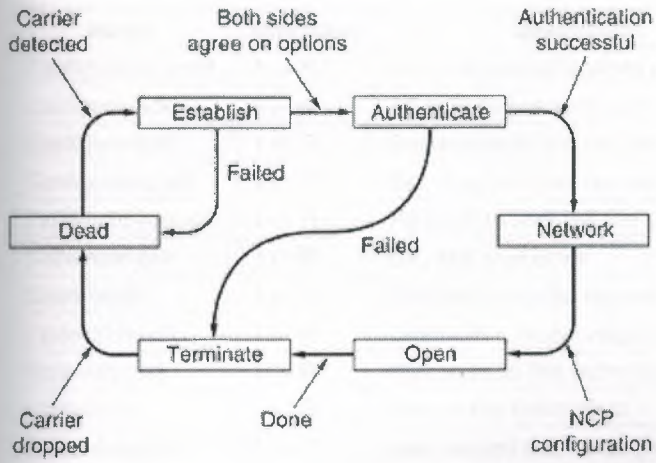
The Payload field is variable length, up to some negotiated maximum. If the length is not negotiated using LCP during line setup, a default length of 1500 bytes is used. Padding may follow the payload if need be.

After the Payload field comes the Checksum field, which is normally 2 bytes, but a 4-byte checksum can be negotiated.

In summary, PPP is a multiprotocol framing mechanism suitable for use over modems, HDLC bit-serial lines, SONET, and other physical layers. It supports error detection, option negotiation, header compression, and, optionally, reliable transmission using an HDLC-type frame format.

Let us now turn from the PPP frame format to the way lines are brought up and down. The (simplified) diagram of Fig. 2-10 shows the phases that a line goes through when it is brought up, used, and taken down again. This sequence applies both to modem connections and to router-router connections.

*Figure 2-10. A simplified phase diagram for bringing a line up and down.*




The protocol starts with the line in the DEAD state, which means that no physical layer carrier is present and no physical layer connection exists. After physical connection is established, the line moves to ESTABLISH. At that point LCP option negotiation begins, which, if successful, leads to AUTHENTICATE. Now the two parties can check on each other's identities if desired. When the NETWORK phase is entered, the appropriate NCP protocol is invoked to configure the network layer. If the configuration is successful, OPEN is reached and data transport can take place. When data transport is finished, the line moves into the TERMINATE phase, and from there, back to DEAD when the carrier is dropped.

LCP negotiates data link protocol options during the ESTABLISH phase. The LCP protocol is not actually concerned with the options themselves, but with the mechanism for negotiation. It provides a way for the initiating process to make a proposal and for the responding process to accept or reject it, in whole or in part. It also provides a way for the two processes to test the line quality to see if they consider it good enough to set up a connection. Finally, the LCP protocol also allows lines to be taken down when they are no longer needed.

Eleven types of LCP frames are defined in RFC 1661. These are listed in Fig. 2-11. The four Configure- types allow the initiator (I) to propose option values and the responder (R) to accept or reject them. In the latter case, the responder can make an alternative proposal or announce that it is not willing to negotiate certain options at all. The options being negotiated and their proposed values are part of the LCP frames.

*Figure 2-11. The LCP frame types.*

| Name | Direction | Description |
|---|---|---|
| Configure-request | I → R | List of proposed options and values |
| Configure-ack | I ← R | All options are accepted |
| Configure-nak | I ← R | Some options are not accepted |
| Configure-reject | I ← R | Some options are not negotiable |
| Terminate-request | I → R | Request to shut the line down |
| Terminate-ack | I ← R | OK, line shut down |
| Code-reject | I ← R | Unknown request received |
| Protocol-reject | I ← R | Unknown protocol requested |
| Echo-request | I → R | Please send this frame back |
| Echo-reply | I ← R | Here is the frame back |
| Discard-request | I → R | Just discard this frame (for testing) |

The Terminate- codes shut a line down when it is no longer needed. The Code-reject and Protocol-reject codes indicate that the responder got something that it does not understand. This situation could mean that an undetected transmission error has occurred, but more likely it means that the initiator and responder are running different versions of the LCP protocol. The Echo- types are used to test the line quality. Finally, Discard-request help debugging. If either end is having trouble getting bits onto the wire, the programmer can use this type for testing. If it manages to get through, the receiver just throws it away, rather than taking some other action that might confuse the person doing the testing.

The options that can be negotiated include setting the maximum payload size for data frames, enabling authentication and choosing a protocol to use, enabling line-quality monitoring during normal operation, and selecting various header compression options.

### *2.2.3 The Network Layer*

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

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

More generally, the quality of service provided (delay, transit time, jitter, etc.) is also a network layer issue.

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

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

### *2.2.4 The Transport Layer*

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

The transport layer also determines what type of service to provide to the session layer, and, ultimately, to the users of the network. The most popular type of transport connection is an error-free point-to-point channel that delivers messages or bytes in the order in which they were sent. However, other possible kinds of transport service are the transporting of isolated messages, with no guarantee about the order of delivery, and the broadcasting of messages to multiple destinations. The type of service is determined when the connection is established. (As an aside, an error-free channel is impossible to achieve; what people really mean by this term is that the error rate is low enough to ignore in practice.)

The transport layer is a true end-to-end layer, all the way from the source to the destination. In other words, a program on the source machine carries on a conversation with a similar program on the destination machine, using the message headers and control messages. In the lower layers, the protocols are between each machine and its immediate neighbors, and not between the ultimate source and destination machines, which may be separated by many routers. The difference between layers 1 through 3, which are chained, and layers 4 through 7, which are end-to-end, is illustrated in Fig. 2-1.

### *2.2.5 The Session Layer*

The session layer allows users on different machines to establish sessions between them. Sessions offer various services, including dialog control (keeping track of whose turn it is to transmit), token management (preventing two parties from attempting the same critical operation at the same time), and synchronization (checkpointing long transmissions to allow them to continue from where they were after a crash).

### *2.2.6 The Presentation Layer*

Unlike lower layers, which are mostly concerned with moving bits around, the presentation layer is concerned with the syntax and semantics of the information transmitted. In order to make it possible for computers with different data representations to communicate, the data structures to be exchanged can be defined in an abstract way, along with a standard encoding to be used "on the wire." The presentation layer manages these abstract data structures and allows higher-level data structures (e.g., banking records), to be defined and exchanged.

### *2.2.7 The Application Layer*

The application layer contains a variety of protocols that are commonly needed by users. One widely-used application protocol is HTTP (HyperText Transfer Protocol), which is the basis for the World Wide Web. When a browser wants a Web page, it sends the name of the page it wants to the server using HTTP. The server then sends the page back. Other application protocols are used for file transfer, electronic mail, and network news.

## 2.3 The TCP/IP Reference Model

Let us now turn from the OSI reference model to the reference model used in the grandparent of all wide area computer networks, the ARPANET, and its successor, the worldwide Internet. Although we will give a brief history of the ARPANET later, it is useful to mention a few key aspects of it now. The ARPANET was a research network sponsored by the DoD (U.S. Department of Defense). It eventually connected hundreds of universities and government installations, using leased telephone lines. When satellite and radio networks were added later, the existing protocols had trouble interworking with them, so a new reference architecture was needed. Thus, the ability to connect multiple networks in a seamless way was one of the major design goals from the very beginning. This architecture later became known as the

TCP/IP Reference Model, after its two primary protocols. It was first defined in (Cerf and Kahn, 1974). A later perspective is given in (Leiner et al., 1985). The design philosophy behind the model is discussed in (Clark, 1988).

Given the DoD's worry that some of its precious hosts, routers, and internetwork gateways might get blown to pieces at a moment's notice, another major goal was that the network be able to survive loss of subnet hardware, with existing conversations not being broken off. In other words, DoD wanted connections to remain intact as long as the source and destination machines were functioning, even if some of the machines or transmission lines in between were suddenly put out of operation. Furthermore, a flexible architecture was needed since applications with divergent requirements were envisioned, ranging from transferring files to real-time speech transmission.

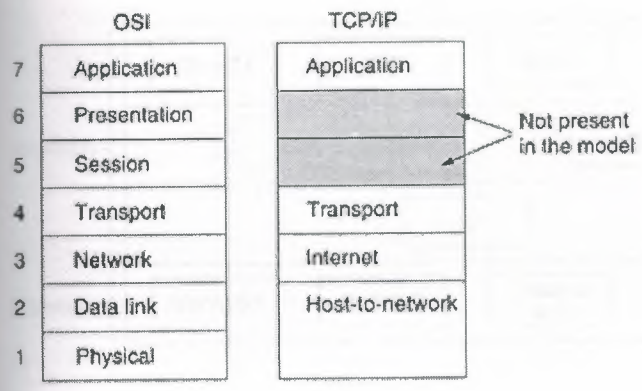
### *2.3.1 The Internet Layer*

All these requirements led to the choice of a packet-switching network based on a connectionless internetwork layer. This layer, called the internet layer, is the linchpin that holds the whole architecture together. Its job is to permit hosts to inject packets into any network and have them travel independently to the destination (potentially on a different network). They may even arrive in a different order than they were sent, in which case it is the job of higher layers to rearrange them, if in-order delivery is desired. Note that "internet" is used here in a generic sense, even though this layer is present in the Internet.

The analogy here is with the (snail) mail system. A person can drop a sequence of international letters into a mail box in one country, and with a little luck, most of them will be delivered to the correct address in the destination country. Probably the letters will travel through one or more international mail gateways along the way, but this is transparent to the users. Furthermore, that each country (i.e., each network) has its own stamps, preferred envelope sizes, and delivery rules is hidden from the users.

The internet layer defines an official packet format and protocol called IP (Internet Protocol). The job of the internet layer is to deliver IP packets where they are supposed to go. Packet routing is clearly the major issue here, as is avoiding congestion. For these reasons, it is reasonable to say that the TCP/IP internet layer is similar in functionality to the OSI network layer. Figure 2-12 shows this correspondence.

*Figure 2-12. The TCP/IP reference model.*

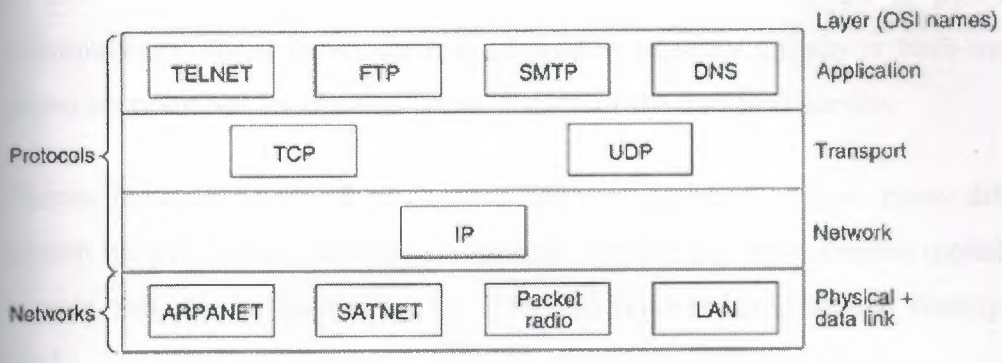



### *2.3.2 The Transport Layer*

The layer above the internet layer in the TCP/IP model is now usually called the transport layer. It is designed to allow peer entities on the source and destination hosts to carry on a conversation, just as in the OSI transport layer. Two end-to-end transport protocols have been defined here. The first one, TCP (Transmission Control Protocol), is a reliable connection-oriented protocol that allows a byte stream originating on one machine to be delivered without error on any other machine in the internet. It fragments the incoming byte stream into discrete messages and passes each one on to the internet layer. At the destination, the receiving TCP process reassembles the received messages into the output stream. TCP also handles flow control to make sure a fast sender cannot swamp a slow receiver with more messages than it can handle.

The second protocol in this layer, UDP (User Datagram Protocol), is an unreliable, connectionless protocol for applications that do not want TCP's sequencing or flow control and wish to provide their own. It is also widely used for one-shot, client-server-type request-reply queries and applications in which prompt delivery is more important than accurate delivery, such as transmitting speech or video. The relation of IP, TCP, and UDP is shown in Fig. 2-13. Since the model was developed, IP has been implemented on many other networks.

*Figure 2-13. Protocols and networks in the TCP/IP model initially.*




### *2.3.3 The Application Layer*

The TCP/IP model does not have session or presentation layers. No need for them was perceived, so they were not included. Experience with the OSI model has proven this view correct: they are of little use to most applications.

On top of the transport layer is the application layer. It contains all the higher-level protocols. The early ones included virtual terminal (TELNET), file transfer (FTP), and electronic mail (SMTP), as shown in Fig. 2-13. The virtual terminal protocol allows a user on one machine to log onto a distant machine and work there. The file transfer protocol provides a way to move data efficiently from one machine to another. Electronic mail was originally just a kind of file transfer, but later a specialized protocol (SMTP) was developed for it. Many other protocols have been added to these over the years: the Domain Name System (DNS) for mapping host names onto their network addresses, NNTP, the protocol for moving USENET news articles around, and HTTP, the protocol for fetching pages on the World Wide Web, and many others.

### *The Host-to-Network Layer*

Below the internet layer is a great void. The TCP/IP reference model does not really say much about what happens here, except to point out that the host has to connect to the network using some protocol so it can send IP packets to it. This protocol is not defined and varies from host to host and network to network. Books and papers about the TCP/IP model rarely discuss it.

## 2.4 A Comparison of the OSI and TCP/IP Reference Models

The OSI and TCP/IP reference models have much in common. Both are based on the concept of a stack of independent protocols. Also, the functionality of the layers is roughly similar.

For example, in both models the layers up through and including the transport layer are there to provide an end-to-end, network-independent transport service to processes wishing to communicate. These layers form the transport provider. Again in both models, the layers above transport are application-oriented users of the transport service.

Despite these fundamental similarities, the two models also have many differences. In this section we will focus on the key differences between the two reference models. It is important to note that we are comparing the reference models here, not the corresponding protocol stacks. The protocols themselves will be discussed later. For an entire book comparing and contrasting TCP/IP and OSI, see (Piscitello and Chapin, 1993).

Three concepts are central to the OSI model:

1. Services.
2. Interfaces.
3. Protocols.

Probably the biggest contribution of the OSI model is to make the distinction between these three concepts explicit. Each layer performs some services for the layer above it. The service definition tells what the layer does, not how entities above it access it or how the layer works. It defines the layer's semantics.

A layer's interface tells the processes above it how to access it. It specifies what the parameters are and what results to expect. It, too, says nothing about how the layer works inside.

Finally, the peer protocols used in a layer are the layer's own business. It can use any protocols it wants to, as long as it gets the job done (i.e., provides the offered services). It can also change them at will without affecting software in higher layers.

These ideas fit very nicely with modern ideas about object-oriented programming. An object, like a layer, has a set of methods (operations) that processes outside the object can invoke. The semantics of these methods define the set of services that the object offers. The methods' parameters and results form the object's interface. The code internal to the object is its protocol and is not visible or of any concern outside the object.

The TCP/IP model did not originally clearly distinguish between service, interface, and protocol, although people have tried to retrofit it after the fact to make it more OSI-like. For example, the only real services offered by the internet layer are SEND IP PACKET and RECEIVE IP PACKET.

As a consequence, the protocols in the OSI model are better hidden than in the TCP/IP model and can be replaced relatively easily as the technology changes. Being able to make such changes is one of the main purposes of having layered protocols in the first place.

The OSI reference model was devised before the corresponding protocols were invented. This ordering means that the model was not biased toward one particular set of protocols, a fact that made it quite general. The downside of this ordering is that the designers did not have much experience with the subject and did not have a good idea of which functionality to put in which layer.

For example, the data link layer originally dealt only with point-to-point networks. When broadcast networks came around, a new sublayer had to be hacked into the model. When people started to build real networks using the OSI model and existing protocols, it was discovered that these networks did not match the required service specifications (wonder of wonders), so convergence sublayers had to be grafted onto the model to provide a place for papering over the differences. Finally, the committee originally expected that each country would have one network, run by the government and using the OSI protocols, so no thought was given to internetworking. To make a long story short, things did not turn out that way.

With TCP/IP the reverse was true: the protocols came first, and the model was really just a description of the existing protocols. There was no problem with the protocols fitting the model. They fit perfectly. The only trouble was that the model did not fit any other protocol stacks. Consequently, it was not especially useful for describing other, non-TCP/IP networks.

Turning from philosophical matters to more specific ones, an obvious difference between the two models is the number of layers: the OSI model has seven layers and the TCP/IP has four layers. Both have (inter)network, transport, and application layers, but the other layers are different.

Another difference is in the area of connectionless versus connection-oriented communication. The OSI model supports both connectionless and connection-oriented

communication in the network layer, but only connection-oriented communication in the transport layer, where it counts (because the transport service is visible to the users). The TCP/IP model has only one mode in the network layer (connectionless) but supports both modes in the transport layer, giving the users a choice. This choice is especially important for simple request-response protocols.

## 2.5 A Critique of the OSI Model and Protocols

Neither the OSI model and its protocols nor the TCP/IP model and its protocols are perfect. Quite a bit of criticism can be, and has been, directed at both of them. In this section and the next one, we will look at some of these criticisms. We will begin with OSI and examine TCP/IP afterward.

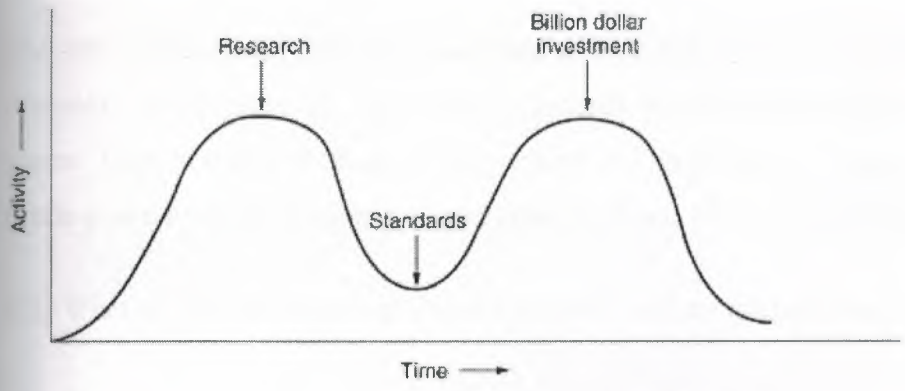
At the time the second edition of this book was published (1989), it appeared to many experts in the field that the OSI model and its protocols were going to take over the world and push everything else out of their way. This did not happen. Why? A look back at some of the lessons may be useful. These lessons can be summarized as:

1. Bad timing.
2. Bad technology.
3. Bad implementations.
4. Bad politics.

### *2.5.1 Bad Timing*

First let us look at reason one: bad timing. The time at which a standard is established is absolutely critical to its success. David Clark of M.I.T. has a theory of standards that he calls the apocalypse of the two elephants, which is illustrated in Fig. 2-14.

*Figure 2-14. The apocalypse of the two elephants.*




This figure shows the amount of activity surrounding a new subject. When the subject is first discovered, there is a burst of research activity in the form of discussions, papers, and meetings. After a while this activity subsides, corporations discover the subject, and the billion-dollar wave of investment hits.

It is essential that the standards be written in the trough in between the two "elephants." If the standards are written too early, before the research is finished, the subject may still be poorly understood; the result is bad standards. If they are written too late, so many companies may have already made major investments in different ways of doing things that the standards are effectively ignored. If the interval between the two elephants is very short (because everyone is in a hurry to get started), the people developing the standards may get crushed.

It now appears that the standard OSI protocols got crushed. The competing TCP/IP protocols were already in widespread use by research universities by the time the OSI protocols appeared. While the billion-dollar wave of investment had not yet hit, the academic market was large enough that many vendors had begun cautiously offering TCP/IP products. When OSI came around, they did not want to support a second protocol stack until they were forced to, so there were no initial offerings. With every company waiting for every other company to go first, no company went first and OSI never happened.

### *2.5.2 Bad Technology*

The second reason that OSI never caught on is that both the model and the protocols are flawed. The choice of seven layers was more political than technical, and two of the layers

(session and presentation) are nearly empty, whereas two other ones (data link and network) are overfull.

The OSI model, along with the associated service definitions and protocols, is extraordinarily complex. When piled up, the printed standards occupy a significant fraction of a meter of paper. They are also difficult to implement and inefficient in operation. In this context, a riddle posed by Paul Mockapetris and cited in (Rose, 1993) comes to mind:

Q1: What do you get when you cross a mobster with an international standard?

A1: Someone who makes you an offer you can't understand.

In addition to being incomprehensible, another problem with OSI is that some functions, such as addressing, flow control, and error control, reappear again and again in each layer. Saltzer et al. (1984), for example, have pointed out that to be effective, error control must be done in the highest layer, so that repeating it over and over in each of the lower layers is often unnecessary and inefficient.

### *2.5.3 Bad Implementations*

Given the enormous complexity of the model and the protocols, it will come as no surprise that the initial implementations were huge, unwieldy, and slow. Everyone who tried them got burned. It did not take long for people to associate "OSI" with "poor quality." Although the products improved in the course of time, the image stuck.

In contrast, one of the first implementations of TCP/IP was part of Berkeley UNIX and was quite good (not to mention, free). People began using it quickly, which led to a large user community, which led to improvements, which led to an even larger community. Here the spiral was upward instead of downward.

### *2.5.4 Bad Politics*

On account of the initial implementation, many people, especially in academia, thought of TCP/IP as part of UNIX, and UNIX in the 1980s in academia was not unlike parenthood (then incorrectly called motherhood) and apple pie.

OSI, on the other hand, was widely thought to be the creature of the European telecommunication ministries, the European Community, and later the U.S. Government. This belief was only partly true, but the very idea of a bunch of government bureaucrats trying to shove a technically inferior standard down the throats of the poor researchers and programmers down in the trenches actually developing computer networks did not help much. Some people viewed this development in the same light as IBM announcing in the 1960s that PL/I was the language of the future, or DoD correcting this later by announcing that it was actually Ada.

### 2.6 A Critique of the TCP/IP Reference Model

The TCP/IP model and protocols have their problems too. First, the model does not clearly distinguish the concepts of service, interface, and protocol. Good software engineering practice requires differentiating between the specification and the implementation, something that OSI does very carefully, and TCP/IP does not. Consequently, the TCP/IP model is not much of a guide for designing new networks using new technologies.

Second, the TCP/IP model is not at all general and is poorly suited to describing any protocol stack other than TCP/IP. Trying to use the TCP/IP model to describe Bluetooth, for example, is completely impossible.

Third, the host-to-network layer is not really a layer at all in the normal sense of the term as used in the context of layered protocols. It is an interface (between the network and data link layers). The distinction between an interface and a layer is crucial, and one should not be sloppy about it.

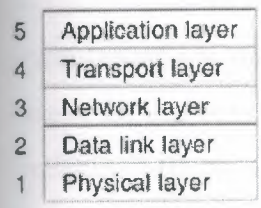
Fourth, the TCP/IP model does not distinguish (or even mention) the physical and data link layers. These are completely different. The physical layer has to do with the transmission characteristics of copper wire, fiber optics, and wireless communication. The data link layer's job is to delimit the start and end of frames and get them from one side to the other with the desired degree of reliability. A proper model should include both as separate layers. The TCP/IP model does not do this.

Finally, although the IP and TCP protocols were carefully thought out and well implemented, many of the other protocols were ad hoc, generally produced by a couple of graduate students hacking away until they got tired. The protocol implementations were then distributed free,

which resulted in their becoming widely used, deeply entrenched, and thus hard to replace. Some of them are a bit of an embarrassment now. The virtual terminal protocol, TELNET, for example, was designed for a ten-character per second mechanical Teletype terminal. It knows nothing of graphical user interfaces and mice. Nevertheless, 25 years later, it is still in widespread use.

In summary, despite its problems, the OSI model (minus the session and presentation layers) has proven to be exceptionally useful for discussing computer networks. In contrast, the OSI protocols have not become popular. The reverse is true of TCP/IP: the model is practically nonexistent, but the protocols are widely used. Since computer scientists like to have their cake and eat it, too, in this book we will use a modified OSI model but concentrate primarily on the TCP/IP and related protocols, as well as newer ones such as 802, SONET, and Bluetooth. In effect, we will use the hybrid model of Fig. 2-15 as the framework for this book.

***Figure 2-15. The hybrid reference model to be used in this book.***




## CHAPTER THREE:EXAMPLE NETWORKS

### 3.1 Example Networks

The subject of computer networking covers many different kinds of networks, large and small, well known and less well known. They have different goals, scales, and technologies. In the following sections, we will look at some examples, to get an idea of the variety one finds in the area of computer networking.

We will start with the Internet, probably the best known network, and look at its history, evolution, and technology. Then we will consider ATM, which is often used within the core of large (telephone) networks. Technically, it is quite different from the Internet, contrasting nicely with it. Next we will introduce Ethernet, the dominant local area network. Finally, we will look at IEEE 802.11, the standard for wireless LANs.

## 3.2 The Internet

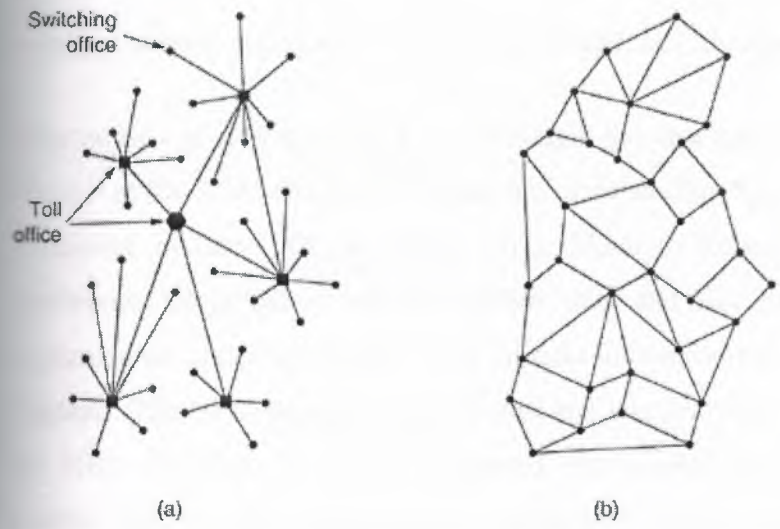
The Internet is not a network at all, but a vast collection of different networks that use certain common protocols and provide certain common services. It is an unusual system in that it was not planned by anyone and is not controlled by anyone. To better understand it, let us start from the beginning and see how it has developed and why. For a wonderful history of the Internet, John Naughton's (2000) book is highly recommended. It is one of those rare books that is not only fun to read, but also has 20 pages of ibid.'s and op. cit.'s for the serious historian. Some of the material below is based on this book.

Of course, countless technical books have been written about the Internet and its protocols as well. For more information, see, for example, (Maufer, 1999).

### *3.2.1 The ARPANET*

The story begins in the late 1950s. At the height of the Cold War, the DoD wanted a command-and-control network that could survive a nuclear war. At that time, all military communications used the public telephone network, which was considered vulnerable. The reason for this belief can be gleaned from Fig. 3-1(a). Here the black dots represent telephone switching offices, each of which was connected to thousands of telephones. These switching offices were, in turn, connected to higher-level switching offices (toll offices), to form a national hierarchy with only a small amount of redundancy. The vulnerability of the system was that the destruction of a few key toll offices could fragment the system into many isolated islands.

*Figure 3-1. (a) Structure of the telephone system. (b) Baran's proposed distributed switching system.*




Around 1960, the DoD awarded a contract to the RAND Corporation to find a solution. One of its employees, Paul Baran, came up with the highly distributed and fault-tolerant design of Fig. 3-1(b). Since the paths between any two switching offices were nowmuch longer than analog signals could travel without distortion, Baran proposed using digital packet-switching technology throughout the system. Baran wrote several reports for the DoD describing his ideas in detail. Officials at the Pentagon liked the concept and asked AT&T, then the U.S. national telephone monopoly, to build a prototype. AT&T dismissed Baran's ideas out of hand. The biggest and richest corporation in the world was not about to allow some young whippersnapper tell it how to build a telephone system. They said Baran's network could not be built and the idea was killed.

Several years went by and still the DoD did not have a better command-and-control system. To understand what happened next, we have to go back to October 1957, when the Soviet Union beat the U.S. into space with the launch of the first artificial satellite, Sputnik. When President Eisenhower tried to find out who was asleep at the switch, he was appalled to find the Army, Navy, and Air Force squabbling over the Pentagon's research budget. His immediate response was to create a single defense research organization, ARPA, the Advanced Research Projects Agency. ARPA had no scientists or laboratories; in fact, it had nothing more than an office and a small (by Pentagon standards) budget. It did its work by issuing grants and contracts to universities and companies whose ideas looked promising to it.

For the first few years, ARPA tried to figure out what its mission should be, but in 1967, the attention of ARPA's then director, Larry Roberts, turned to networking. He contacted various experts to decide what to do. One of them, Wesley Clark, suggested building a packet-switched subnet, giving each host its own router, as illustrated in Fig. 1-5.

After some initial skepticism, Roberts bought the idea and presented a somewhat vague paper about it at the ACM SIGOPS Symposium on Operating System Principles held in Gatlinburg, Tennessee in late 1967 (Roberts, 1967). Much to Roberts' surprise, another paper at the conference described a similar system that had not only been designed but actually implemented under the direction of Donald Davies at the National Physical Laboratory in England. The NPL system was not a national system (it just connected several computers on the NPL campus), but it demonstrated that packet switching could be made to work. Furthermore, it cited Baran's now discarded earlier work. Roberts came away from Gatlinburg determined to build what later became known as the ARPANET.

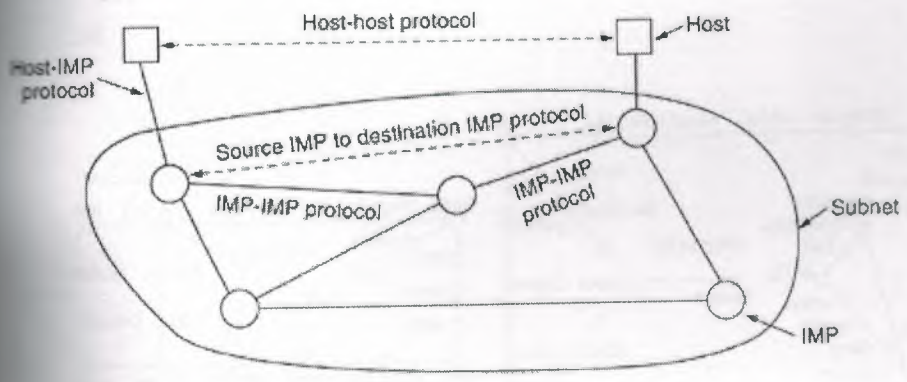
The subnet would consist of minicomputers called IMPs (Interface Message Processors) connected by 56-kbps transmission lines. For high reliability, each IMP would be connected to at least two other IMPs. The subnet was to be a datagram subnet, so if some lines and IMPs were destroyed, messages could be automatically rerouted along alternative paths.

Each node of the network was to consist of an IMP and a host, in the same room, connected by a short wire. A host could send messages of up to 8063 bits to its IMP, which would then break these up into packets of at most 1008 bits and forward them independently toward the destination. Each packet was received in its entirety before being forwarded, so the subnet was the first electronic store-and-forward packet-switching network.

ARPA then put out a tender for building the subnet. Twelve companies bid for it. Afte evaluating all the proposals, ARPA selected BBN, a consulting firm in Cambridge, Massachusetts, and in December 1968, awarded it a contract to build the subnet and write the subnet software. BBN chose to use specially modified Honeywell DDP-316 minicomputers with 12K 16-bit words of core memory as the IMPs. The IMPs did not have disks, since moving parts were considered unreliable. The IMPs were interconnected by 56-kbps lines leased from telephone companies. Although 56 kbps is now the choice of teenagers who cannot afford ADSL or cable, it was then the best money could buy.

The software was split into two parts: subnet and host. The subnet software consisted of the IMP end of the host-IMP connection, the IMP-IMP protocol, and a source IMP to destination IMP protocol designed to improve reliability. The original ARPANET design is shown in Fig. 3-2.

*Figure 3-2. The original ARPANET design.*

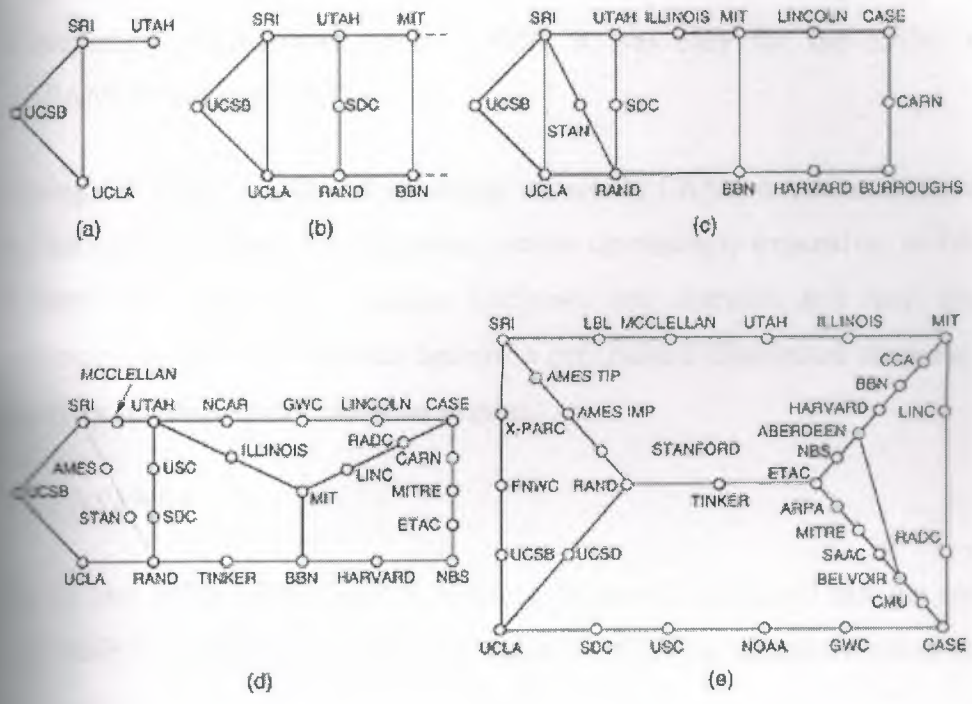



Outside the subnet, software was also needed, namely, the host end of the host-IMP connection, the host-host protocol, and the application software. It soon became clear that BBN felt that when it had accepted a message on a host-IMP wire and placed it on the host-IMP wire at the destination, its job was done.

Roberts had a problem: the hosts needed software too. To deal with it, he convened a meeting of network researchers, mostly graduate students, at Snowbird, Utah, in the summer of 1969. The graduate students expected some network expert to explain the grand design of the network and its software to them and then to assign each of them the job of writing part of it. They were astounded when there was no network expert and no grand design. They had to figure out what to do on their own.

Nevertheless, somehow an experimental network went on the air in December 1969 with four nodes: at UCLA, UCSB, SRI, and the University of Utah. These four were chosen because all had a large number of ARPA contracts, and all had different and completely incompatible host computers (just to make it more fun). The network grew quickly as more IMPs were delivered and installed; it soon spanned the United States. Figure 3-3 shows how rapidly the ARPANET grew in the first 3 years.

*Figure 3-3. Growth of the ARPANET. (a) December 1969. (b) July 1970. (c) March 1971. (d) April 1972. (e) September 1972.*




In addition to helping the fledgling ARPANET grow, ARPA also funded research on the use of satellite networks and mobile packet radio networks. In one now famous demonstration, a truck driving around in California used the packet radio network to send messages to SRI, which were then forwarded over the ARPANET to the East Coast, where they were shipped to University College in London over the satellite network. This allowed a researcher in the truck to use a computer in London while driving around in California.

This experiment also demonstrated that the existing ARPANET protocols were not suitable for running over multiple networks. This observation led to more research on protocols, culminating with the invention of the TCP/IP model and protocols (Cerf and Kahn, 1974). TCP/IP was specifically designed to handle communication over internetworks, something becoming increasingly important as more and more networks were being hooked up to the ARPANET.

To encourage adoption of these new protocols, ARPA awarded several contracts to BBN and the University of California at Berkeley to integrate them into Berkeley UNIX. Researchers at Berkeley developed a convenient program interface to the network (sockets) and wrote many application, utility, and management programs to make networking easier.

The timing was perfect. Many universities had just acquired a second or third VAX computer and a LAN to connect them, but they had no networking software. When 4.2BSD came along, with TCP/IP, sockets, and many network utilities, the complete package was adopted immediately. Furthermore, with TCP/IP, it was easy for the LANs to connect to the ARPANET, and many did.

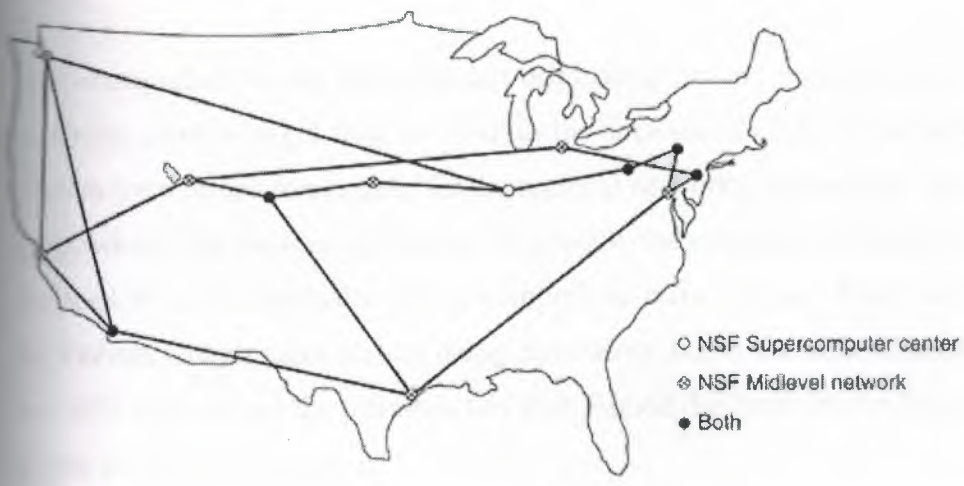
During the 1980s, additional networks, especially LANs, were connected to the ARPANET. As the scale increased, finding hosts became increasingly expensive, so DNS (Domain Name System) was created to organize machines into domains and map host names onto IP addresses. Since then, DNS has become a generalized, distributed database system for storing a variety of information related to naming.

### *3.2.2 NSFNET*

By the late 1970s, NSF (the U.S. National Science Foundation) saw the enormous impact the ARPANET was having on university research, allowing scientists across the country to share data and collaborate on research projects. However, to get on the ARPANET, a university had to have a research contract with the DoD, which many did not have. NSF's response was to design a successor to the ARPANET that would be open to all university research groups. To have something concrete to start with, NSF decided to build a backbone network to connect its six supercomputer centers, in San Diego, Boulder, Champaign, Pittsburgh, Ithaca, and Princeton. Each supercomputer was given a little brother, consisting of an LSI-11 microcomputer called a fuzzball. The fuzzballs were connected with 56-kbps leased lines and formed the subnet, the same hardware technology as the ARPANET used. The software technology was different however: the fuzzballs spoke TCP/IP right from the start, making it the first TCP/IP WAN.

NSF also funded some (eventually about 20) regional networks that connected to the backbone to allow users at thousands of universities, research labs, libraries, and museums to access any of the supercomputers and to communicate with one another. The complete network, including the backbone and the regional networks, was called NSFNET. It connected to the ARPANET through a link between an IMP and a fuzzball in the Carnegie-Mellon machine room. The first NSFNET backbone is illustrated in Fig. 3-4.

*Figure 3-4. The NSFNET backbone in 1988.*




NSFNET was an instantaneous success and was overloaded from the word go. NSF immediately began planning its successor and awarded a contract to the Michigan-based MERIT consortium to run it. Fiber optic channels at 448 kbps were leased from MCI (since merged with WorldCom) to provide the version 2 backbone. IBM PC-RTs were used as routers. This, too, was soon overwhelmed, and by 1990, the second backbone was upgraded to 1.5 Mbps.

As growth continued, NSF realized that the government could not continue financing networking forever. Furthermore, commercial organizations wanted to join but were forbidden by NSF's charter from using networks NSF paid for. Consequently, NSF encouraged MERIT, MCI, and IBM to form a nonprofit corporation, ANS (Advanced Networks and Services), as the first step along the road to commercialization. In 1990, ANS took over NSFNET and upgraded the 1.5-Mbps links to 45 Mbps to form ANSNET. This network operated for 5 years and was then sold to America Online. But by then, various companies were offering commercial IP service and it was clear the government should now get out of the networking business.

To ease the transition and make sure every regional network could communicate with every other regional network, NSF awarded contracts to four different network operators to establish a NAP (Network Access Point). These operators were PacBell (San Francisco), Ameritech (Chicago), MFS (Washington, D.C.), and Sprint (New York City, where for NAP purposes, Pennsauken, New Jersey counts as New York City). Every network operator that

wanted to provide backbone service to the NSF regional networks had to connect to all the NAPs.

This arrangement meant that a packet originating on any regional network had a choice of backbone carriers to get from its NAP to the destination's NAP. Consequently, the backbone carriers were forced to compete for the regional networks' business on the basis of service and price, which was the idea, of course. As a result, the concept of a single default backbone was replaced by a commercially-driven competitive infrastructure. Many people like to criticize the Federal Government for not being innovative, but in the area of networking, it was DoD and NSF that created the infrastructure that formed the basis for the Internet and then handed it over to industry to operate.

During the 1990s, many other countries and regions also built national research networks, often patterned on the ARPANET and NSFNET. These included EuropaNET and EBONE in Europe, which started out with 2-Mbps lines and then upgraded to 34-Mbps lines. Eventually, the network infrastructure in Europe was handed over to industry as well.

### *3.2.3 Internet Usage*

The number of networks, machines, and users connected to the ARPANET grew rapidly after TCP/IP became the only official protocol on January 1, 1983. When NSFNET and the ARPANET were interconnected, the growth became exponential. Many regional networks joined up, and connections were made to networks in Canada, Europe, and the Pacific.

Sometime in the mid-1980s, people began viewing the collection of networks as an internet, and later as the Internet, although there was no official dedication with some politician breaking a bottle of champagne over a fuzzball.

The glue that holds the Internet together is the TCP/IP reference model and TCP/IP protocol stack. TCP/IP makes universal service possible and can be compared to the adoption of standard gauge by the railroads in the 19th century or the adoption of common signaling protocols by all the telephone companies.

What does it actually mean to be on the Internet? Our definition is that a machine is on the Internet if it runs the TCP/IP protocol stack, has an IP address, and can send IP packets to all the other machines on the Internet. The mere ability to send and receive electronic mail is not

enough, since e-mail is gatewayed to many networks outside the Internet. However, the issue is clouded somewhat by the fact that millions of personal computers can call up an Internet service provider using a modem, be assigned a temporary IP address, and send IP packets to other Internet hosts. It makes sense to regard such machines as being on the Internet for as long as they are connected to the service provider's router.

Traditionally (meaning 1970 to about 1990), the Internet and its predecessors had four main applications:

1. **E-mail.** The ability to compose, send, and receive electronic mail has been around since the early days of the ARPANET and is enormously popular. Many people get dozens of messages a day and consider it their primary way of interacting with the outside world, far outdistancing the telephone and snail mail. E-mail programs are available on virtually every kind of computer these days.
2. **News.** Newsgroups are specialized forums in which users with a common interest can exchange messages. Thousands of newsgroups exist, devoted to technical and nontechnical topics, including computers, science, recreation, and politics. Each newsgroup has its own etiquette, style, and customs, and woe betide anyone violating them.
3. **Remote login.** Using the telnet, rlogin, or ssh programs, users anywhere on the Internet can log on to any other machine on which they have an account.
4. **File transfer.** Using the FTP program, users can copy files from one machine on the Internet to another. Vast numbers of articles, databases, and other information are available this way.

Up until the early 1990s, the Internet was largely populated by academic, government, and industrial researchers. One new application, the WWW (World Wide Web) changed all that and brought millions of new, nonacademic users to the net. This application, invented by CERN physicist Tim Berners-Lee, did not change any of the underlying facilities but made them easier to use. Together with the Mosaic browser, written by Marc Andreessen at the National Center for Supercomputer Applications in Urbana, Illinois, the WWW made it possible for a site to set up a number of pages of information containing text, pictures, sound, and even video, with embedded links to other pages. By clicking on a link, the user is suddenly transported to the page pointed to by that link. For example, many companies have a

home page with entries pointing to other pages for product information, price lists, sales, technical support, communication with employees, stockholder information, and more.

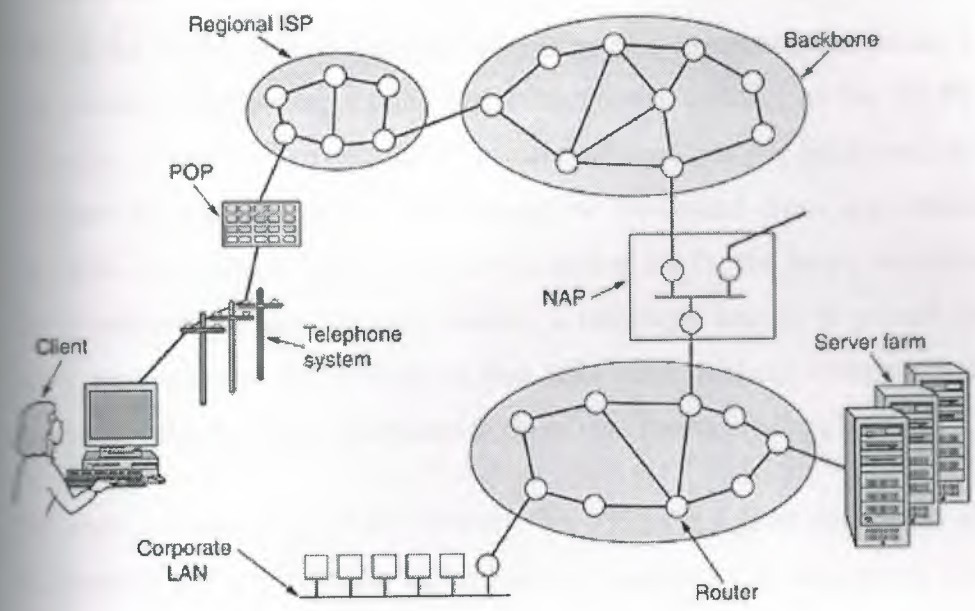
Numerous other kinds of pages have come into existence in a very short time, including maps, stock market tables, library card catalogs, recorded radio programs, and even a page pointing to the complete text of many books whose copyrights have expired (Mark Twain, Charles Dickens, etc.). Many people also have personal pages (home pages).

Much of this growth during the 1990s was fueled by companies called ISPs (Internet Service Providers). These are companies that offer individual users at home the ability to call up one of their machines and connect to the Internet, thus gaining access to e-mail, the WWW, and other Internet services. These companies signed up tens of millions of new users a year during the late 1990s, completely changing the character of the network from an academic and military playground to a public utility, much like the telephone system. The number of Internet users now is unknown, but is certainly hundreds of millions worldwide and will probably hit 1 billion fairly soon.

### *3.2.4 Architecture of the Internet*

In this section we will attempt to give a brief overview of the Internet today. Due to the many mergers between telephone companies (telcos) and ISPs, the waters have become muddied and it is often hard to tell who is doing what. Consequently, this description will be of necessity somewhat simpler than reality. The big picture is shown in Fig. 3-5.

**Figure 3-5 Overview of the Internet.**




A good place to start is with a client at home. Let us assume our client calls his or her ISP over a dial-up telephone line, as shown in Fig. 3-5. The modem is a card within the PC that converts the digital signals the computer produces to analog signals that can pass unhindered over the telephone system. These signals are transferred to the ISP's POP (Point of Presence), where they are removed from the telephone system and injected into the ISP's regional network. From this point on, the system is fully digital and packet switched. If the ISP is the local telco, the POP will probably be located in the telephone switching office where the telephone wire from the client terminates. If the ISP is not the local telco, the POP may be a few switching offices down the road.

The ISP's regional network consists of interconnected routers in the various cities the ISP serves. If the packet is destined for a host served directly by the ISP, the packet is delivered to the host. Otherwise, it is handed over to the ISP's backbone operator.

At the top of the food chain are the major backbone operators, companies like AT&T and Sprint. They operate large international backbone networks, with thousands of routers connected by high-bandwidth fiber optics. Large corporations and hosting services that run server farms (machines that can serve thousands of Web pages per second) often connect directly to the backbone. Backbone operators encourage this direct connection by renting space in what are called carrier hotels, basically equipment racks in the same room as the router to allow short, fast connections between server farms and the backbone.

If a packet given to the backbone is destined for an ISP or company served by the backbone, it is sent to the closest router and handed off there. However, many backbones, of varying sizes, exist in the world, so a packet may have to go to a competing backbone. To allow packets to hop between backbones, all the major backbones connect at the NAPs discussed earlier. Basically, a NAP is a room full of routers, at least one per backbone. A LAN in the room connects all the routers, so packets can be forwarded from any backbone to any other backbone. In addition to being interconnected at NAPs, the larger backbones have numerous direct connections between their routers, a technique known as private peering. One of the many paradoxes of the Internet is that ISPs who publicly compete with one another for customers often privately cooperate to do private peering (Metz, 2001).

This ends our quick tour of the Internet. We will have a great deal to say about the individual components and their design, algorithms, and protocols in subsequent chapters. Also worth mentioning in passing is that some companies have interconnected all their existing internal networks, often using the same technology as the Internet. These intranets are typically accessible only within the company but otherwise work the same way as the Internet.

### 3.3 Connection-Oriented Networks: X.25, Frame Relay, and ATM

Since the beginning of networking, a war has been going on between the people who support connectionless (i.e., datagram) subnets and the people who support connection-oriented subnets. The main proponents of the connectionless subnets come from the ARPANET/Internet community. Remember that DoD's original desire in funding and building the ARPANET was to have a network that would continue functioning even after multiple direct hits by nuclear weapons wiped out numerous routers and transmission lines. Thus, fault tolerance was high on their priority list; billing customers was not. This approach led to a connectionless design in which every packet is routed independently of every other packet. As a consequence, if some routers go down during a session, no harm is done as long as the system can reconfigure itself dynamically so that subsequent packets can find some route to the destination, even if it is different from that which previous packets used.

The connection-oriented camp comes from the world of telephone companies. In the telephone system, a caller must dial the called party's number and wait for a connection before talking or sending data. This connection setup establishes a route through the telephone

system that is maintained until the call is terminated. All words or packets follow the same route. If a line or switch on the path goes down, the call is aborted.

This property is precisely what the DoD did not like about it.

Why do the telephone companies like it then? There are two reasons:

1. Quality of service.
2. Billing.

By setting up a connection in advance, the subnet can reserve resources such as buffer space and router CPU capacity. If an attempt is made to set up a call and insufficient resources are available, the call is rejected and the caller gets a kind of busy signal. In this way, once a connection has been set up, the connection will get good service. With a connectionless network, if too many packets arrive at the same router at the same moment, the router will choke and probably lose packets. The sender will eventually notice this and resend them, but the quality of service will be jerky and unsuitable for audio or video unless the network is very lightly loaded. Needless to say, providing adequate audio quality is something telephone companies care about very much, hence their preference for connections.

The second reason the telephone companies like connection-oriented service is that they are accustomed to charging for connect time. When you make a long distance call (or even a local call outside North America) you are charged by the minute. When networks came around, they just automatically gravitated toward a model in which charging by the minute was easy to do. If you have to set up a connection before sending data, that is when the billing clock starts running. If there is no connection, they cannot charge for it.

Ironically, maintaining billing records is very expensive. If a telephone company were to adopt a flat monthly rate with unlimited calling and no billing or record keeping, it would probably save a huge amount of money, despite the increased calling this policy would generate. Political, regulatory, and other factors weigh against doing this, however. Interestingly enough, flat rate service exists in other sectors. For example, cable TV is billed at a flat rate per month, no matter how many programs you watch. It could have been designed with pay-per-view as the basic concept, but it was not, due in part to the expense of billing (and given the quality of most television, the embarrassment factor cannot be totally

discounted either). Also, many theme parks charge a daily admission fee for unlimited rides, in contrast to traveling carnivals, which charge by the ride.

That said, it should come as no surprise that all networks designed by the telephone industry have had connection-oriented subnets. What is perhaps surprising, is that the Internet is also drifting in that direction, in order to provide a better quality of service for audio and video. But now let us examine some connection-oriented networks.

### *3.3.1 X.25 and Frame Relay*

Our first example of a connection-oriented network is X.25, which was the first public data network. It was deployed in the 1970s at a time when telephone service was a monopoly everywhere and the telephone company in each country expected there to be one data network per country—theirs. To use X.25, a computer first established a connection to the remote computer, that is, placed a telephone call. This connection was given a connection number to be used in data transfer packets (because multiple connections could be open at the same time). Data packets were very simple, consisting of a 3-byte header and up to 128 bytes of data. The header consisted of a 12-bit connection number, a packet sequence number, an acknowledgement number, and a few miscellaneous bits. X.25 networks operated for about a decade with mixed success.

In the 1980s, the X.25 networks were largely replaced by a new kind of network called frame relay. The essence of frame relay is that it is a connection-oriented network with no error control and no flow control. Because it was connection-oriented, packets were delivered in order (if they were delivered at all). The properties of in-order delivery, no error control, and no flow control make frame relay akin to a wide area LAN. Its most important application is interconnecting LANs at multiple company offices. Frame relay enjoyed a modest success and is still in use in places today.

### *3.3.2 Asynchronous Transfer Mode*

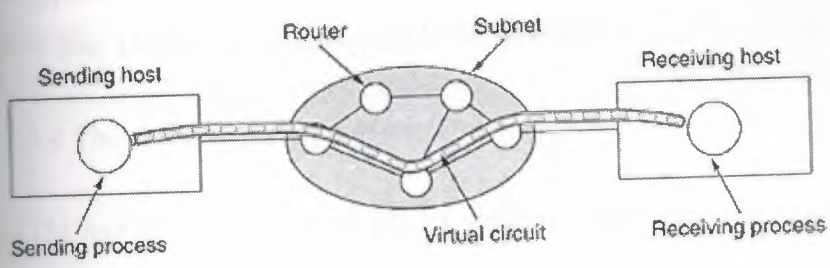
Yet another, and far more important, connection-oriented network is ATM (Asynchronous Transfer Mode). The reason for the somewhat strange name is that in the telephone system, most transmission is synchronous (closely tied to a clock), and ATM is not.

ATM was designed in the early 1990s and launched amid truly incredible hype (Ginsburg, 1996; Goralski, 1995; Ibe, 1997; Kim et al., 1994; and Stallings, 2000). ATM was going to solve all the world's networking and telecommunications problems by merging voice, data, cable television, telex, telegraph, carrier pigeon, tin cans connected by strings, tom-toms, smoke signals, and everything else into a single integrated system that could do everything for everyone. It did not happen. In large part, the problems were similar to those we described earlier concerning OSI, that is, bad timing, technology, implementation, and politics. Having just beaten back the telephone companies in round 1, many in the Internet community saw ATM as Internet versus the Telcos: the Sequel. But it really was not, and this time around even diehard datagram fanatics were aware that the Internet's quality of service left a lot to be desired. To make a long story short, ATM was much more successful than OSI, and it is now widely used deep within the telephone system, often for moving IP packets. Because it is now mostly used by carriers for internal transport, users are often unaware of its existence, but it is definitely alive and well.

### *3.3.3 ATM Virtual Circuits*

Since ATM networks are connection-oriented, sending data requires first sending a packet to set up the connection. As the setup packet wends its way through the subnet, all the routers on the path make an entry in their internal tables noting the existence of the connection and reserving whatever resources are needed for it. Connections are often called virtual circuits, in analogy with the physical circuits used within the telephone system. Most ATM networks also support permanent virtual circuits, which are permanent connections between two (distant) hosts. They are similar to leased lines in the telephone world. Each connection, temporary or permanent, has a unique connection identifier. A virtual circuit is illustrated in Fig. 3-6.

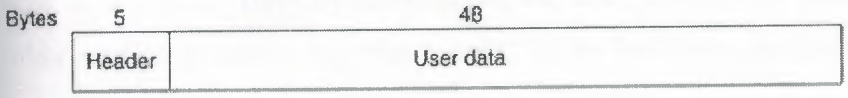
***Figure 3-6. A virtual circuit.***




Once a connection has been established, either side can begin transmitting data. The basic idea behind ATM is to transmit all information in small, fixed-size packets called cells. The

cells are 53 bytes long, of which 5 bytes are header and 48 bytes are payload, as shown in Fig. 3-7. Part of the header is the connection identifier, so the sending and receiving hosts and all the intermediate routers can tell which cells belong to which connections. This information allows each router to know how to route each incoming cell. Cell routing is done in hardware, at high speed. In fact, the main argument for having fixed-size cells is that it is easy to build hardware routers to handle short, fixed-length cells. Variable-length IP packets have to be routed by software, which is a slower process. Another plus of ATM is that the hardware can be set up to copy one incoming cell to multiple output lines, a property that is required for handling a television program that is being broadcast to many receivers. Finally, small cells do not block any line for very long, which makes guaranteeing quality of service easier.

***Figure 3-7. An ATM cell.***



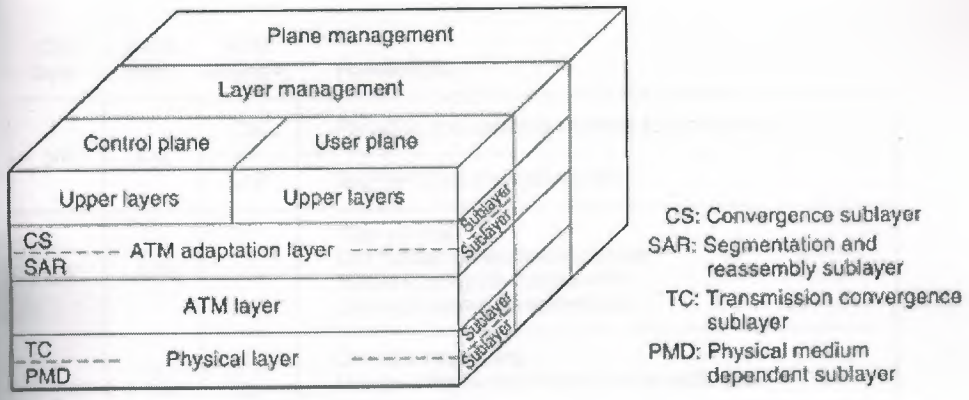

All cells follow the same route to the destination. Cell delivery is not guaranteed, but their order is. If cells 1 and 2 are sent in that order, then if both arrive, they will arrive in that order, never first 2 then 1. But either or both of them can be lost along the way. It is up to higher protocol levels to recover from lost cells. Note that although this guarantee is not perfect, it is better than what the Internet provides. There packets can not only be lost, but delivered out of order as well. ATM, in contrast, guarantees never to deliver cells out of order.

ATM networks are organized like traditional WANs, with lines and switches (routers). The most common speeds for ATM networks are 155 Mbps and 622 Mbps, although higher speeds are also supported. The 155-Mbps speed was chosen because this is about what is needed to transmit high definition television. The exact choice of 155.52 Mbps was made for compatibility with AT&T's SONET transmission system. The 622 Mbps speed was chosen so that four 155-Mbps channels could be sent over it.

### *3.3.4 The ATM Reference Model*

ATM has its own reference model, different from the OSI model and also different from the TCP/IP model. This model is shown in Fig. 3-8. It consists of three layers, the physical, ATM, and ATM adaptation layers, plus whatever users want to put on top of that.

*Figure 3-8. The ATM reference model.*




The physical layer deals with the physical medium: voltages, bit timing, and various other issues. ATM does not prescribe a particular set of rules but instead says that ATM cells can be sent on a wire or fiber by themselves, but they can also be packaged inside the payload of other carrier systems. In other words, ATM has been designed to be independent of the transmission medium.

The ATM layer deals with cells and cell transport. It defines the layout of a cell and tells what the header fields mean. It also deals with establishment and release of virtual circuits. Congestion control is also located here.

Because most applications do not want to work directly with cells (although some may), a layer above the ATM layer has been defined to allow users to send packets larger than a cell. The ATM interface segments these packets, transmits the cells individually, and reassembles them at the other end. This layer is the AAL (ATM Adaptation Layer).

Unlike the earlier two-dimensional reference models, the ATM model is defined as being three-dimensional, as shown in Fig. 3-8. The user plane deals with data transport, flow control, error correction, and other user functions. In contrast, the control plane is concerned with connection management. The layer and plane management functions relate to resource management and interlayer coordination.

The physical and AAL layers are each divided into two sublayers, one at the bottom that does the work and a convergence sublayer on top that provides the proper interface to the layer above it. The functions of the layers and sublayers are given in Fig. 3-9.

*Figure 3-9. The ATM layers and sublayers, and their functions.*

| OSI layer | ATM layer | ATM sublayer | Functionality |
|---|---|---|---|
| 3/4 | AAL | CS | Providing the standard interface (convergence) |
| | | SAR | Segmentation and reassembly |
| 2/3 | ATM | | Flow control<br>Cell header generation/extraction<br>Virtual circuit/path management<br>Cell multiplexing/demultiplexing |
| 2 | Physical | TC | Cell rate decoupling<br>Header checksum generation and verification<br>Cell generation<br>Packing/unpacking cells from the enclosing envelope<br>Frame generation |
| 1 | | PMD | Bit timing<br>Physical network access |

The PMD (Physical Medium Dependent) sublayer interfaces to the actual cable. It moves the bits on and off and handles the bit timing. For different carriers and cables, this layer will be different.

The other sublayer of the physical layer is the TC (Transmission Convergence) sublayer. When cells are transmitted, the TC layer sends them as a string of bits to the PMD layer. Doing this is easy. At the other end, the TC sublayer gets a pure incoming bit stream from the PMD sublayer. Its job is to convert this bit stream into a cell stream for the ATM layer. It handles all the issues related to telling where cells begin and end in the bit stream. In the ATM model, this functionality is in the physical layer. In the OSI model and in pretty much all other networks, the job of framing, that is, turning a raw bit stream into a sequence of frames or cells, is the data link layer's task.

As we mentioned earlier, the ATM layer manages cells, including their generation and transport. Most of the interesting aspects of ATM are located here. It is a mixture of the OSI data link and network layers; it is not split into sublayers.

The AAL layer is split into a SAR (Segmentation And Reassembly) sublayer and a CS (Convergence Sublayer). The lower sublayer breaks up packets into cells on the transmission side and puts them back together again at the destination. The upper sublayer makes it

possible to have ATM systems offer different kinds of services to different applications (e.g., file transfer and video on demand have different requirements concerning error handling, timing, etc.).

As it is probably mostly downhill for ATM from now on, we will not discuss it further in this book. Nevertheless, since it has a substantial installed base, it will probably be around for at least a few more years. For more information about ATM, see (Dobrowski and Grise, 2001; and Gadecki and Heckart, 1997).

### 3.4 Ethernet

Both the Internet and ATM were designed for wide area networking. However, many companies, universities, and other organizations have large numbers of computers that must be connected. This need gave rise to the local area network. In this section we will say a little bit about the most popular LAN, Ethernet.

The story starts out in pristine Hawaii in the early 1970s. In this case, "pristine" can be interpreted as "not having a working telephone system." While not being interrupted by the phone all day long makes life more pleasant for vacationers, it did not make life more pleasant for researcher Norman Abramson and his colleagues at the University of Hawaii who were trying to connect users on remote islands to the main computer in Honolulu. Stringing their own cables under the Pacific Ocean was not in the cards, so they looked for a different solution.

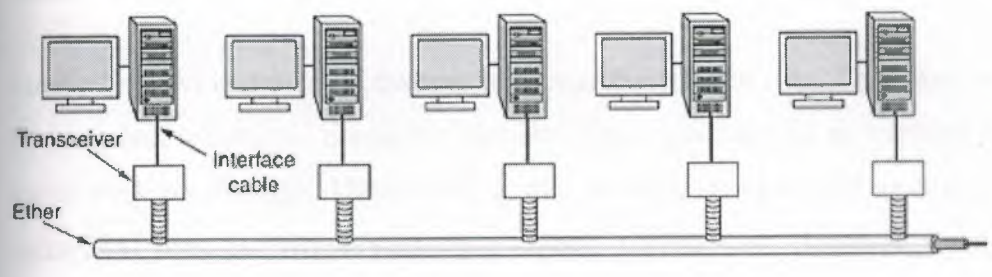
The one they found was short-range radios. Each user terminal was equipped with a small radio having two frequencies: upstream (to the central computer) and downstream (from the central computer). When the user wanted to contact the computer, it just transmitted a packet containing the data in the upstream channel. If no one else was transmitting at that instant, the packet probably got through and was acknowledged on the downstream channel. If there was contention for the upstream channel, the terminal noticed the lack of acknowledgement and tried again. Since there was only one sender on the downstream channel (the central computer), there were never collisions there. This system, called ALOHANET, worked fairly well under conditions of low traffic but bogged down badly when the upstream traffic was heavy.

About the same time, a student named Bob Metcalfe got his bachelor's degree at M.I.T. and then moved up the river to get his Ph.D. at Harvard. During his studies, he was exposed to Abramson's work. He became so interested in it that after graduating from Harvard, he decided to spend the summer in Hawaii working with Abramson before starting work at Xerox PARC (Palo Alto Research Center). When he got to PARC, he saw that the researchers there had designed and built what would later be called personal computers. But the machines were isolated. Using his knowledge of Abramson's work, he, together with his colleague David Boggs, designed and implemented the first local area network (Metcalfe and Boggs, 1976).

They called the system Ethernet after the luminiferous ether, through which electromagnetic radiation was once thought to propagate. (When the 19th century British physicist James Clerk Maxwell discovered that electromagnetic radiation could be described by a wave equation, scientists assumed that space must be filled with some ethereal medium in which the radiation was propagating. Only after the famous Michelson-Morley experiment in 1887 did physicists discover that electromagnetic radiation could propagate in a vacuum.)

The transmission medium here was not a vacuum, but a thick coaxial cable (the ether) up to 2.5 km long (with repeaters every 500 meters). Up to 256 machines could be attached to the system via transceivers screwed onto the cable. A cable with multiple machines attached to it in parallel is called a multidrop cable. The system ran at 2.94 Mbps. A sketch of its architecture is given in Fig. 3-10. Ethernet had a major improvement over ALOHANET: before transmitting, a computer first listened to the cable to see if someone else was already transmitting. If so, the computer held back until the current transmission finished. Doing so avoided interfering with existing transmissions, giving a much higher efficiency. ALOHANET did not work like this because it was impossible for a terminal on one island to sense the transmission of a terminal on a distant island. With a single cable, this problem does not exist.

*Figure 3-10. Architecture of the original Ethernet.*




Despite the computer listening before transmitting, a problem still arises: what happens if two or more computers all wait until the current transmission completes and then all start at once? The solution is to have each computer listen during its own transmission and if it detects interference, jam the ether to alert all senders. Then back off and wait a random time before retrying. If a second collision happens, the random waiting time is doubled, and so on, to spread out the competing transmissions and give one of them a chance to go first.

The Xerox Ethernet was so successful that DEC, Intel, and Xerox drew up a standard in 1978 for a 10-Mbps Ethernet, called the DIX standard. With two minor changes, the DIX standard became the IEEE 802.3 standard in 1983.

Unfortunately for Xerox, it already had a history of making seminal inventions (such as the personal computer) and then failing to commercialize on them, a story told in Fumbling the Future (Smith and Alexander, 1988). When Xerox showed little interest in doing anything with Ethernet other than helping standardize it, Metcalfe formed his own company, 3Com, to sell Ethernet adapters for PCs. It has sold over 100 million of them.

Ethernet continued to develop and is still developing. New versions at 100 Mbps, 1000 Mbps, and still higher have come out. Also the cabling has improved, and switching and other features have been added.

In passing, it is worth mentioning that Ethernet (IEEE 802.3) is not the only LAN standard. The committee also standardized a token bus (802.4) and a token ring (802.5). The need for three more-or-less incompatible standards has little to do with technology and everything to do with politics. At the time of standardization, General Motors was pushing a LAN in which the topology was the same as Ethernet (a linear cable) but computers took turns in transmitting by passing a short packet called a token from computer to computer. A computer could only send if it possessed the token, thus avoiding collisions. General Motors announced

that this scheme was essential for manufacturing cars and was not prepared to budge from this position. This announcement notwithstanding, 802.4 has basically vanished from sight.

Similarly, IBM had its own favorite: its proprietary token ring. The token was passed around the ring and whichever computer held the token was allowed to transmit before putting the token back on the ring. Unlike 802.4, this scheme, standardized as 802.5, is still in use at some IBM sites, but virtually nowhere outside of IBM sites. However, work is progressing on a gigabit version (802.5v), but it seems unlikely that it will ever catch up with Ethernet. In short, there was a war between Ethernet, token bus, and token ring, and Ethernet won, mostly because it was there first and the challengers were not as good.

### 6.9 Wireless LANs: 802.11

Almost as soon as notebook computers appeared, many people had a dream of walking into an office and magically having their notebook computer be connected to the Internet. Consequently, various groups began working on ways to accomplish this goal. The most practical approach is to equip both the office and the notebook computers with short-range radio transmitters and receivers to allow them to communicate. This work rapidly led to wireless LANs being marketed by a variety of companies.

The trouble was that no two of them were compatible. This proliferation of standards meant that a computer equipped with a brand X radio would not work in a room equipped with a brand Y base station. Finally, the industry decided that a wireless LAN standard might be a good idea, so the IEEE committee that standardized the wired LANs was given the task of drawing up a wireless LAN standard. The standard it came up with was named 802.11. A common slang name for it is WiFi. It is an important standard and deserves respect, so we will call it by its proper name, 802.11.

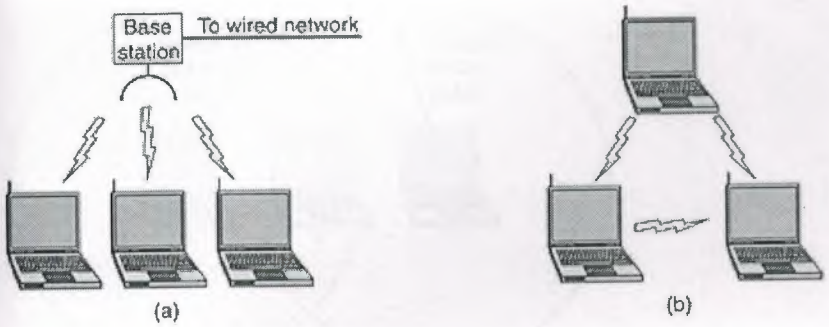
The proposed standard had to work in two modes:

1. In the presence of a base station.
2. In the absence of a base station.

In the former case, all communication was to go through the base station, called an access point in 802.11 terminology. In the latter case, the computers would just send to one another directly. This mode is now sometimes called ad hoc networking. A typical example is two or

more people sitting down together in a room not equipped with a wireless LAN and having their computers just communicate directly. The two modes are illustrated in Fig. 3-11.

***Figure 3-11. (a) Wireless networking with a base station. (b) Ad hoc networking.***




The first decision was the easiest: what to call it. All the other LAN standards had numbers like 802.1, 802.2, 802.3, up to 802.10, so the wireless LAN standard was dubbed 802.11. The rest was harder.

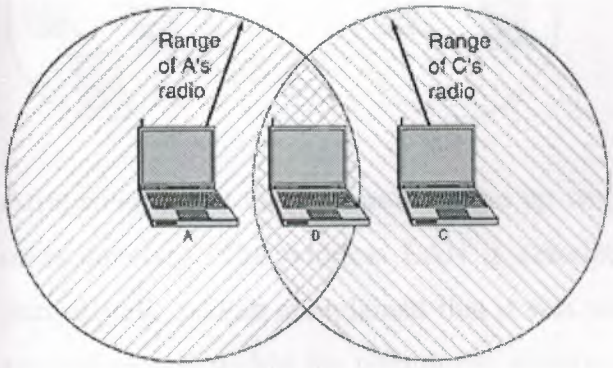
In particular, some of the many challenges that had to be met were: finding a suitable frequency band that was available, preferably worldwide; dealing with the fact that radio signals have a finite range; ensuring that users' privacy was maintained; taking limited battery life into account; worrying about human safety (do radio waves cause cancer?); understanding the implications of computer mobility; and finally, building a system with enough bandwidth to be economically viable.

At the time the standardization process started (mid-1990s), Ethernet had already come to dominate local area networking, so the committee decided to make 802.11 compatible with Ethernet above the data link layer. In particular, it should be possible to send an IP packet over the wireless LAN the same way a wired computer sent an IP packet over Ethernet. Nevertheless, in the physical and data link layers, several inherent differences with Ethernet exist and had to be dealt with by the standard.

First, a computer on Ethernet always listens to the ether before transmitting. Only if the ether is idle does the computer begin transmitting. With wireless LANs, that idea does not work so well. To see why, examine Fig. 3-12. Suppose that computer A is transmitting to computer B, but the radio range of A's transmitter is too short to reach computer C. If C wants to transmit

to B it can listen to the ether before starting, but the fact that it does not hear anything does not mean that its transmission will succeed. The 802.11 standard had to solve this problem.

***Figure 3-12. The range of a single radio may not cover the entire system.***

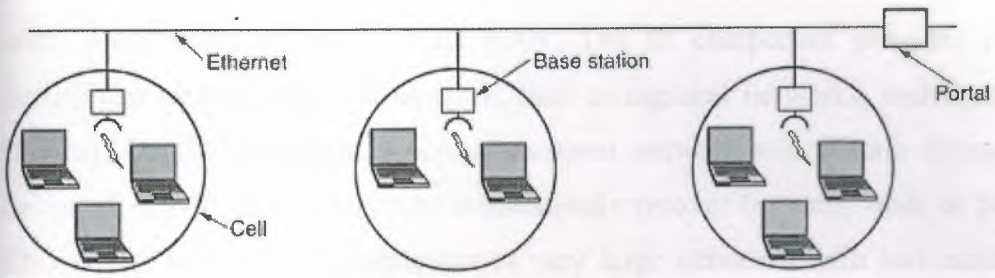



The second problem that had to be solved is that a radio signal can be reflected off solid objects, so it may be received multiple times (along multiple paths). This interference results in what is called multipath fading.

The third problem is that a great deal of software is not aware of mobility. For example, many word processors have a list of printers that users can choose from to print a file. When the computer on which the word processor runs is taken into a new environment, the built-in list of printers becomes invalid.

The fourth problem is that if a notebook computer is moved away from the ceiling-mounted base station it is using and into the range of a different base station, some way of handing it off is needed. Although this problem occurs with cellular telephones, it does not occur with Ethernet and needed to be solved. In particular, the network envisioned consists of multiple cells, each with its own base station, but with the base stations connected by Ethernet, as shown in Fig. 3-13. From the outside, the entire system should look like a single Ethernet. The connection between the 802.11 system and the outside world is called a portal.

*Figure 3-13. A multicell 802.11 network.*




After some work, the committee came up with a standard in 1997 that addressed these and other concerns. The wireless LAN it described ran at either 1 Mbps or 2 Mbps. Almost immediately, people complained that it was too slow, so work began on faster standards. A split developed within the committee, resulting in two new standards in 1999. The 802.11a standard uses a wider frequency band and runs at speeds up to 54 Mbps. The 802.11b standard uses the same frequency band as 802.11, but uses a different modulation technique to achieve 11 Mbps. Some people see this as psychologically important since 11 Mbps is faster than the original wired Ethernet. It is likely that the original 1-Mbps 802.11 will die off quickly, but it is not yet clear which of the new standards will win out.

To make matters even more complicated than they already were, the 802 committee has come up with yet another variant, 802.11g, which uses the modulation technique of 802.11a but the frequency band of 802.11b.

That 802.11 is going to cause a revolution in computing and Internet access is now beyond any doubt. Airports, train stations, hotels, shopping malls, and universities are rapidly installing it. Even upscale coffee shops are installing 802.11 so that the assembled yuppies can surf the Web while drinking their lattes. It is likely that 802.11 will do to the Internet what notebook computers did to computing: make it mobile.

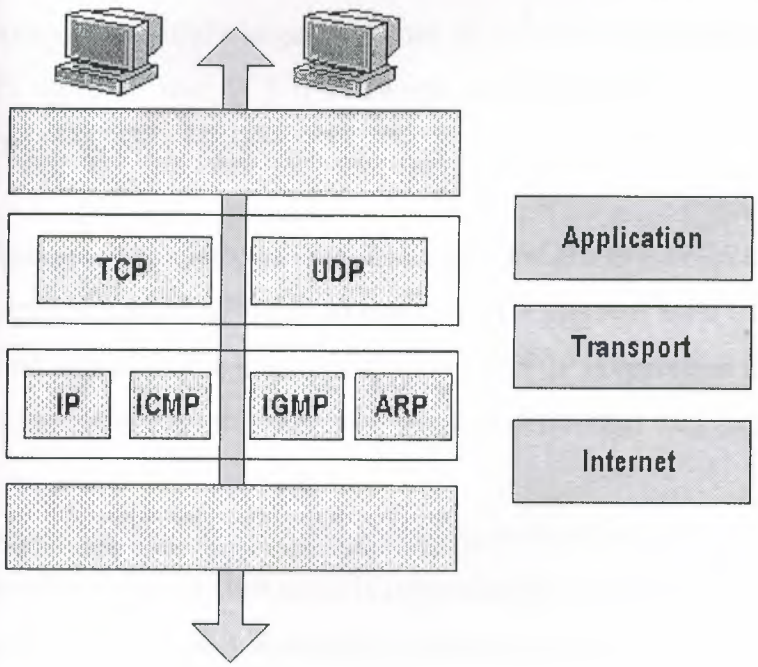
## CHAPTER FOUR:CONTENTS OF TCP/IP

### 4.1 Introduction to TCP/IP

Summary: TCP and IP were developed by a Department of Defense (DOD) research project to connect a number different networks designed by different vendors into a network of networks (the "Internet"). It was initially successful because it delivered a few basic services

that everyone needs (file transfer, electronic mail, remote logon) across a very large number of client and server systems. Several computers in a small department can use TCP/IP (along with other protocols) on a single LAN. The IP component provides routing from the department to the enterprise network, then to regional networks, and finally to the global Internet. On the battlefield a communications network will sustain damage, so the DOD designed TCP/IP to be robust and automatically recover from any node or phone line failure. This design allows the construction of very large networks with less central management. However, because of the automatic recovery, network problems can go undiagnosed and uncorrected for long periods of time.

***Figure 4-1.TCP/IP is composed of layers:***




- **IP** - is responsible for moving packet of data from node to node. IP forwards each packet based on a four byte destination address (the IP number). The Internet authorities assign ranges of numbers to different organizations. The organizations assign groups of their numbers to departments. IP operates on gateway machines that move data from department to organization to region and then around the world.
- **TCP** - is responsible for verifying the correct delivery of data from client to server. Data can be lost in the intermediate network. TCP adds support to detect errors or lost data and to trigger retransmission until the data is correctly and completely received.

- **Sockets** - is a name given to the package of subroutines that provide access to TCP/IP on most systems.

## 4.2 What Is TCP/IP?

TCP/IP is a set of **protocols** that enable communication between computers. There was a time when it was not important for computers to communicate with each other. There was no need for a common protocol. But as computers became networked, the need arose for computers to agree on certain protocols.**Protocols** Rules or standards that govern communications.

Today, a **network administrator** can choose from many protocols, but the TCP/IP protocol is the most widely used. Part of the reason is that TCP/IP is the protocol of choice on the Internet-the world's largest network. If you want a computer to communicate on the Internet, it'll have to use TCP/IP.**Network administrator** A person who installs, monitors, and troubleshoots a network.

When multiple protocols work together, the group is collectively known as a protocol suite or protocol stack. TCP/IP is an example of a protocol suite (it describes multiple protocols that work together).The implementation of TCP/IP is described as a protocol stack. Both terms are used interchangeably, yet their definitions vary slightly.

Another reason for TCP/IP's popularity is that it is compatible with almost every computer in the world. The TCP/IP stack is supported by current versions of all the major operating systems and network operating systems-including Windows 95/98, Windows NT, Windows 2000, Windows XP, Linux, Unix, and NetWare.

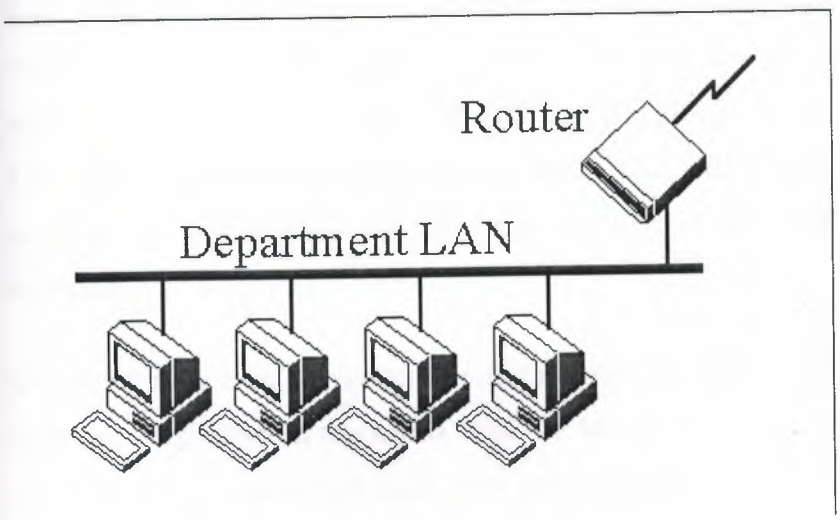
Unlike proprietary protocols developed by hardware and software vendors to make their equipment work, TCP/IP enjoys support from a variety of hardware and software vendors. Examples of companies that have products that work with TCP/IP include Microsoft, Novell, IBM, Apple, and Red Hat. Many other companies also support the TCP/IP protocol suite. TCP/IP is sometimes referred to as "the language of the Internet." In addition to being the official language of the Internet, TCP/IP is also the official language of many smaller networks. For all the computers that are attached to the Internet to communicate effectively, they must agree on a language. Just like every human language has certain rules so that the

people involved in the conversation understand what the other is saying, a computer language needs a set of rules so that computers can effectively communicate. Some of the rules of a language that computers use to communicate include determining when to send data and when to receive data.

### 4.2.1 Network of Lowest Bidders

The Army puts out a bid on a computer and DEC wins the bid. The Air Force puts out a bid and IBM wins. The Navy bid is won by Unisys. Then the President decides to invade Grenada and the armed forces discover that their computers cannot talk to each other. The DOD must build a "network" out of systems each of which, by law, was delivered by the lowest bidder on a single contract.

***Figure 4-2.Computers connected to each other with a Router***




The Internet Protocol was developed to create a Network of Networks (the "Internet"). Individual machines are first connected to a LAN (Ethernet or Token Ring). TCP/IP shares the LAN with other uses (a Novell file server, Windows for Workgroups peer systems). One device provides the TCP/IP connection between the LAN and the rest of the world.

To insure that all types of systems from all vendors can communicate, TCP/IP is absolutely standardized on the LAN. However, larger networks based on long distances and phone lines are more volatile. In the US, many large corporations would wish to reuse large internal networks based on IBM's SNA. In Europe, the national phone companies traditionally standardize on X.25. However, the sudden explosion of high speed microprocessors, fiber

optics, and digital phone systems has created a burst of new options: ISDN, frame relay, FDDI, Asynchronous Transfer Mode (ATM). New technologies arise and become obsolete within a few years. With cable TV and phone companies competing to build the National Information Superhighway, no single standard can govern citywide, nationwide, or worldwide communications.

The original design of TCP/IP as a Network of Networks fits nicely within the current technological uncertainty. TCP/IP data can be sent across a LAN, or it can be carried within an internal corporate SNA network, or it can piggyback on the cable TV service. Furthermore, machines connected to any of these networks can communicate to any other network through gateways supplied by the network vendor.

### 4.2.2 Addresses

Each technology has its own convention for transmitting messages between two machines within the same network. On a LAN, messages are sent between machines by supplying the six byte unique identifier (the "MAC" address). In an SNA network, every machine has Logical Units with their own network address. DECNET, Appletalk, and Novell IPX all have a scheme for assigning numbers to each local network and to each workstation attached to the network.

On top of these local or vendor specific network addresses, TCP/IP assigns a unique number to every workstation in the world. This "IP number" is a four byte value that, by convention, is expressed by converting each byte into a decimal number (0 to 255) and separating the bytes with a period. For example, the PC Lube and Tune server is 130.132.59.234.

An organization begins by sending electronic mail to Hostmaster@INTERNIC.NET requesting assignment of a network number. It is still possible for almost anyone to get assignment of a number for a small "Class C" network in which the first three bytes identify the network and the last byte identifies the individual computer. The author followed this procedure and was assigned the numbers 192.35.91.* for a network of computers at his house. Larger organizations can get a "Class B" network where the first two bytes identify the network and the last two bytes identify each of up to 64 thousand individual workstations. Yale's Class B network is 130.132, so all computers with IP address 130.132.*.* are connected through Yale.

The organization then connects to the Internet through one of a dozen regional or specialized network suppliers. The network vendor is given the subscriber network number and adds it to the routing configuration in its own machines and those of the other major network suppliers.

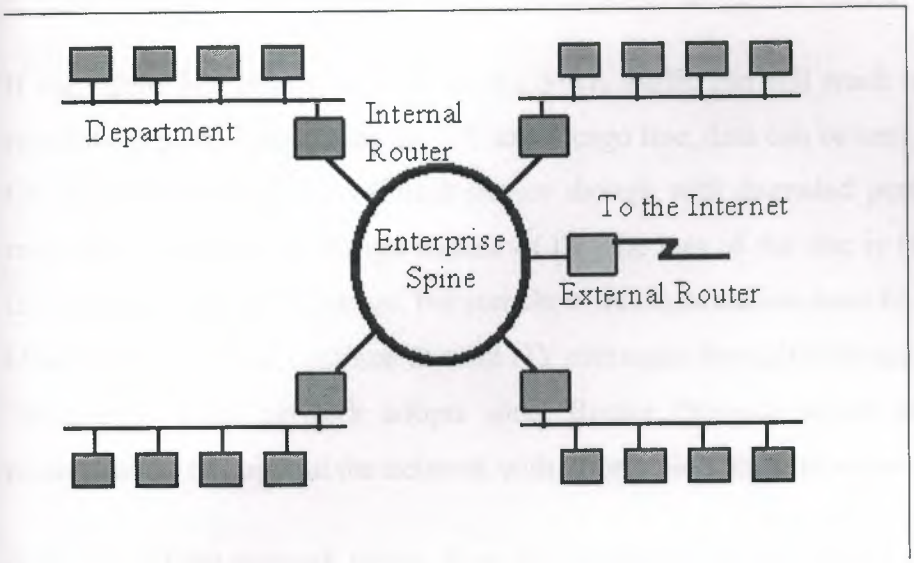
There is no mathematical formula that translates the numbers 192.35.91 or 130.132 into "Yale University" or "New Haven, CT." The machines that manage large regional networks or the central Internet routers managed by the National Science Foundation can only locate these networks by looking each network number up in a table. There are potentially thousands of Class B networks, and millions of Class C networks, but computer memory costs are low, so the tables are reasonable. Customers that connect to the Internet, even customers as large as IBM, do not need to maintain any information on other networks. They send all external data to the regional carrier to which they subscribe, and the regional carrier maintains the tables and does the appropriate routing.

New Haven is in a border state, split 50-50 between the Yankees and the Red Sox. In this spirit, Yale recently switched its connection from the Middle Atlantic regional network to the New England carrier. When the switch occurred, tables in the other regional areas and in the national spine had to be updated, so that traffic for 130.132 was routed through Boston instead of New Jersey. The large network carriers handle the paperwork and can perform such a switch given sufficient notice. During a conversion period, the university was connected to both networks so that messages could arrive through either path.

### 4.2.3 Subnets

Although the individual subscribers do not need to tabulate network numbers or provide explicit routing, it is convenient for most Class B networks to be internally managed as a much smaller and simpler version of the larger network organizations. It is common to subdivide the two bytes available for internal assignment into a one byte department number and a one byte workstation ID.

*Figure 4-3.Subnetting of networks*




The enterprise network is built using commercially available TCP/IP router boxes. Each router has small tables with 255 entries to translate the one byte department number into selection of a destination Ethernet connected to one of the routers. Messages to the PC Lube and Tune server (130.132.59.234) are sent through the national and New England regional networks based on the 130.132 part of the number. Arriving at Yale, the 59 department ID selects an Ethernet connector in the C& IS building. The 234 selects a particular workstation on that LAN. The Yale network must be updated as new Ethernets and departments are added, but it is not effected by changes outside the university or the movement of machines within the department.

### 4.2.4 A Uncertain Path

Every time a message arrives at an IP router, it makes an individual decision about where to send it next. There is concept of a session with a preselected path for all traffic. Consider a company with facilities in New York, Los Angeles, Chicago and Atlanta. It could build a network from four phone lines forming a loop (NY to Chicago to LA to Atlanta to NY). A message arriving at the NY router could go to LA via either Chicago or Atlanta. The reply could come back the other way.

How does the router make a decision between routes? There is no correct answer. Traffic could be routed by the "clockwise" algorithm (go NY to Atlanta, LA to Chicago). The routers

could alternate, sending one message to Atlanta and the next to Chicago. More sophisticated routing measures traffic patterns and sends data through the least busy link.

If one phone line in this network breaks down, traffic can still reach its destination through a roundabout path. After losing the NY to Chicago line, data can be sent NY to Atlanta to LA to Chicago. This provides continued service though with degraded performance. This kind of recovery is the primary design feature of IP. The loss of the line is immediately detected by the routers in NY and Chicago, but somehow this information must be sent to the other nodes. Otherwise, LA could continue to send NY messages through Chicago, where they arrive at a "dead end." Each network adopts some Router Protocol which periodically updates the routing tables throughout the network with information about changes in route status.

If the size of the network grows, then the complexity of the routing updates will increase as will the cost of transmitting them. Building a single network that covers the entire US would be unreasonably complicated. Fortunately, the Internet is designed as a Network of Networks. This means that loops and redundancy are built into each regional carrier. The regional network handles its own problems and reroutes messages internally. Its Router Protocol updates the tables in its own routers, but no routing updates need to propagate from a regional carrier to the NSF spine or to the other regions (unless, of course, a subscriber switches permanently from one region to another).

### 4.2.5 Undiagnosed Problems

IBM designs its SNA networks to be centrally managed. If any error occurs, it is reported to the network authorities. By design, any error is a problem that should be corrected or repaired. IP networks, however, were designed to be robust. In battlefield conditions, the loss of a node or line is a normal circumstance. Casualties can be sorted out later on, but the network must stay up. So IP networks are robust. They automatically (and silently) reconfigure themselves when something goes wrong. If there is enough redundancy built into the system, then communication is maintained.

In 1975 when SNA was designed, such redundancy would be prohibitively expensive, or it might have been argued that only the Defense Department could afford it. Today, however, simple routers cost no more than a PC. However, the TCP/IP design that, "Errors are normal and can be largely ignored," produces problems of its own.

Data traffic is frequently organized around "hubs," much like airline traffic. One could imagine an IP router in Atlanta routing messages for smaller cities throughout the Southeast. The problem is that data arrives without a reservation. Airline companies experience the problem around major events, like the Super Bowl. Just before the game, everyone wants to fly into the city. After the game, everyone wants to fly out. Imbalance occurs on the network when something new gets advertised. Adam Curry announced the server at "mtv.com" and his regional carrier was swamped with traffic the next day. The problem is that messages come in from the entire world over high speed lines, but they go out to mtv.com over what was then a slow speed phone line.

Occasionally a snow storm cancels flights and airports fill up with stranded passengers. Many go off to hotels in town. When data arrives at a congested router, there is no place to send the overflow. Excess packets are simply discarded. It becomes the responsibility of the sender to retry the data a few seconds later and to persist until it finally gets through. This recovery is provided by the TCP component of the Internet protocol.

TCP was designed to recover from node or line failures where the network propagates routing table changes to all router nodes. Since the update takes some time, TCP is slow to initiate recovery. The TCP algorithms are not tuned to optimally handle packet loss due to traffic congestion. Instead, the traditional Internet response to traffic problems has been to increase the speed of lines and equipment in order to say ahead of growth in demand.

TCP treats the data as a stream of bytes. It logically assigns a sequence number to each byte. The TCP packet has a header that says, in effect, "This packet starts with byte 379642 and contains 200 bytes of data." The receiver can detect missing or incorrectly sequenced packets. TCP acknowledges data that has been received and retransmits data that has been lost. The TCP design means that error recovery is done end-to-end between the Client and Server machine. There is no formal standard for tracking problems in the middle of the network, though each network has adopted some ad hoc tools.

### 4.2.6 Need to Know

There are three levels of TCP/IP knowledge. Those who administer a regional or national network must design a system of long distance phone lines, dedicated routing devices, and very large configuration files. They must know the IP numbers and physical locations of

thousands of subscriber networks. They must also have a formal network monitor strategy to detect problems and respond quickly.

Each large company or university that subscribes to the Internet must have an intermediate level of network organization and expertise. A half dozen routers might be configured to connect several dozen departmental LANs in several buildings. All traffic outside the organization would typically be routed to a single connection to a regional network provider.

However, the end user can install TCP/IP on a personal computer without any knowledge of either the corporate or regional network. Three pieces of information are required:

1. The IP address assigned to this personal computer
2. The part of the IP address (the subnet mask) that distinguishes other machines on the same LAN (messages can be sent to them directly) from machines in other departments or elsewhere in the world (which are sent to a router machine)
3. The IP address of the router machine that connects this LAN to the rest of the world.

In the case of the PCLT server, the IP address is 130.132.59.234. Since the first three bytes designate this department, a "subnet mask" is defined as 255.255.255.0 (255 is the largest byte value and represents the number with all bits turned on). It is a Yale convention (which we recommend to everyone) that the router for each department have station number 1 within the department network. Thus the PCLT router is 130.132.59.1. Thus the PCLT server is configured with the values:

- My IP address: 130.132.59.234
- Subnet mask: 255.255.255.0
- Default router: 130.132.59.1

The subnet mask tells the server that any other machine with an IP address beginning 130.132.59.* is on the same department LAN, so messages are sent to it directly. Any IP address beginning with a different value is accessed indirectly by sending the message through the router at 130.132.59.1 (which is on the departmental LAN).

Additional information is available in self-study courses from SRA (1-800-SRA-1277)

- TCP/IP [34610]

## 4.3 Features of TCP/IP

TCP/IP has been in a use for more than 20 years, and time has proven it to be a tested and stable protocol suite. TCP/IP has many features and benefits. In this section, you will learn about some of the most important ones.

### Support from Vendors

As stated earlier, TCP/IP receives support from many hardware and software vendors. This means that the TCP/IP suite is not tied to the development efforts of a single company. Instead, the choice to use TCP/IP on a network can be based on the purpose of the network and not on the hardware or software that has been purchased.

### Interoperability

One of the major reasons why the TCP/IP suite has gained popularity and acceptance so universally is that it can be installed and used on virtually every platform. For example, using TCP/IP, a Unix host can communicate and transfer data to a DOS host or a Windows host. A **host** is another name for a computer or device on a network. TCP/IP eliminates the crossplatform boundaries.**Host** Any device (such as a workstation, server, mainframe, or printer) on a network or internetwork that has a TCP/IP address.

### Flexibility

TCP/IP is an extremely flexible protocol suite, and in later chapters you will learn about some features that contribute to this flexibility. Examples of TCP/IP's flexibility include the latitude an administrator has in assigning and reassigning addresses. An administrator can automatically or manually assign an IP address to a host, and a TCP/IP host can convert easyto-remember names, such as www.sybex.com, to a TCP/IP address.

### Routability

A limitation of many protocols is their difficulty moving data from one segment of the network to another. TCP/IP is exceptionally well adapted to the process of routing data from one segment of the network to another, or from a host on a network in one part of the world to a host on a network in another part of the world.

In the following sections, you will learn about how these features of TCP/IP grew out of the military's need for a reliable, flexible networking standard.

### 4.4 Design Goals of TCP/IP

TCP/IP has evolved to its current state. The protocols within the TCP/IP suite have been tested, modified, and improved over time. The original TCP/IP protocol suite had several design goals that intended to make it a viable protocol for the large, evolving internetwork. Some of these goals included:

- **Hardware independence** A protocol suite that could be used on a Mac, PC, mainframe, or any other computer.
- **Software independence** A protocol suite that could be used by different software vendors and applications. This would enable a host on one site to communicate with a host on another site, without having the same software configuration.
- **Failure recovery and the ability to handle high error rates** A protocol suite that featured automatic recovery from any dropped or lost data. This protocol must be able to recover from an outage of any host on any part of the network and at any point in a data transfer.
- **Efficient protocol with low overhead** A protocol suite that had a minimal amount of "extra" data moving with the data being transferred. This extra data, called overhead, functions as packaging for the data being transferred and enables the data transmission. Overhead is similar to an envelope used to send a letter, or a box used to send a bigger item-having too much overhead is as efficient as using a large crate to send someone a necklace.
- **Ability to add new networks to the internetwork without service disruption** A protocol suite that enabled new, independent networks to join this network of networks without bringing down the larger internetwork.
- **Routable Data** A protocol suite on which data could make its way through an internetwork of computers to any possible destination. For this to be possible, a single and meaningful addressing scheme must be used so that every computer that is moving the data can compute the best path of every piece of data as it moves through the network.

The TCP/IP porotocol suite has evolved to meet these goals. Throughout this book, you will learn how TCP/IP has met and surpassed these original design goals.

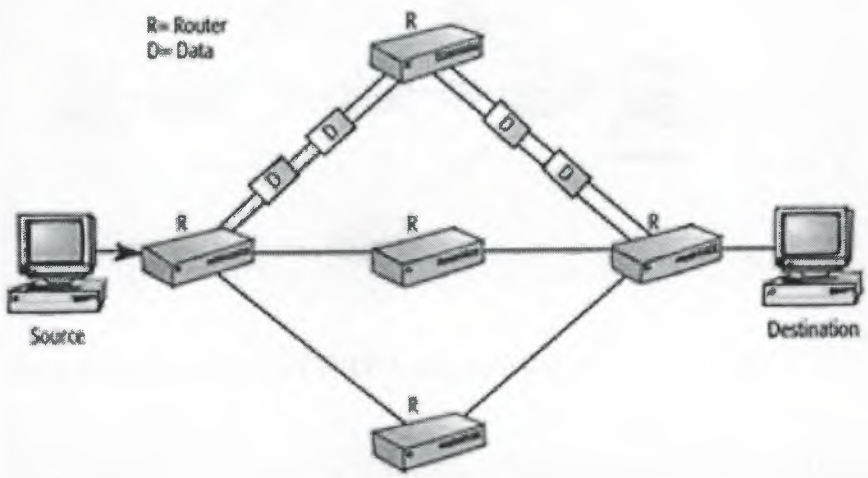
### 4.5 Moving Data across the Network

Creating this new "super network" introduced many new concepts and challenges for the pioneering engineers. One of the most critical issues was how to move data across the network. Older communications protocols relied on a circuit-switched technology. TCP/IP, however, introduced a new way of moving data across a network. The protocol suite set a new standard for communications and data transport by using a packet-switched network.

#### 4.5.1Moving Data on a Circuit-Switched Network

Historically, data has moved through a **circuit-switched network**. In a circuit-switched network, data moves across the same path throughout the entire communication. An example of a circuit-switched network is the telephone system. When you make a telephone call, a single path (also called a circuit) is established between the caller and the recipient. For the rest of the conversation, the voice data keeps moving through the same circuit. If you were to make a call and get a very staticky connection, you would hang up and try again. This way you could get a different circuit, hopefully one with less static. Early network data transmissions followed this type of pathway.

**circuit-switched network** A network on which all data in a communication takes the same path.In the illustration below, notice that although the data could take multiple routes, all the datamoves from the source to the destination along the same path. In a circuit-switched network,data communication moves along a single, established route.

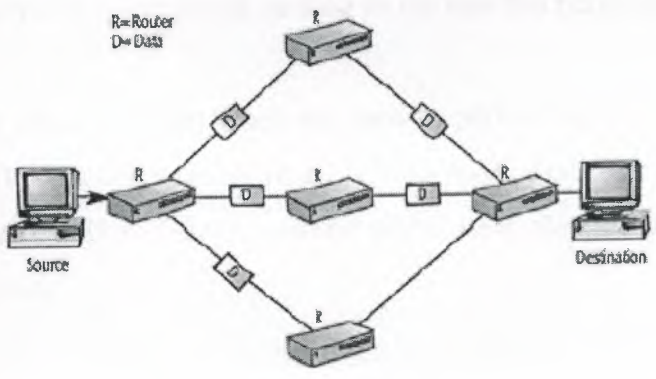


### 4.5.2 Moving Data on a Packet-Switched Network

A circuit-switched network was unacceptable for both the ARPAnet and the Internet. Data had to be able to move through different routes so that if one circuit went down or got staticky, it didn't affect communication on the rest of the network. Instead, data simply would take a different route.

The Internet uses a **packet-switched network**. On a packet-switched network, the computer that is sending the data fragments the data into smaller, more manageable chunks. These chunks are called **packets**. Each packet is then individually addressed and sent to its intended recipient. As the several packets make their way through the network, each packet finds its own way to the receiver. The receiving computer reassembles the packets into the original message.

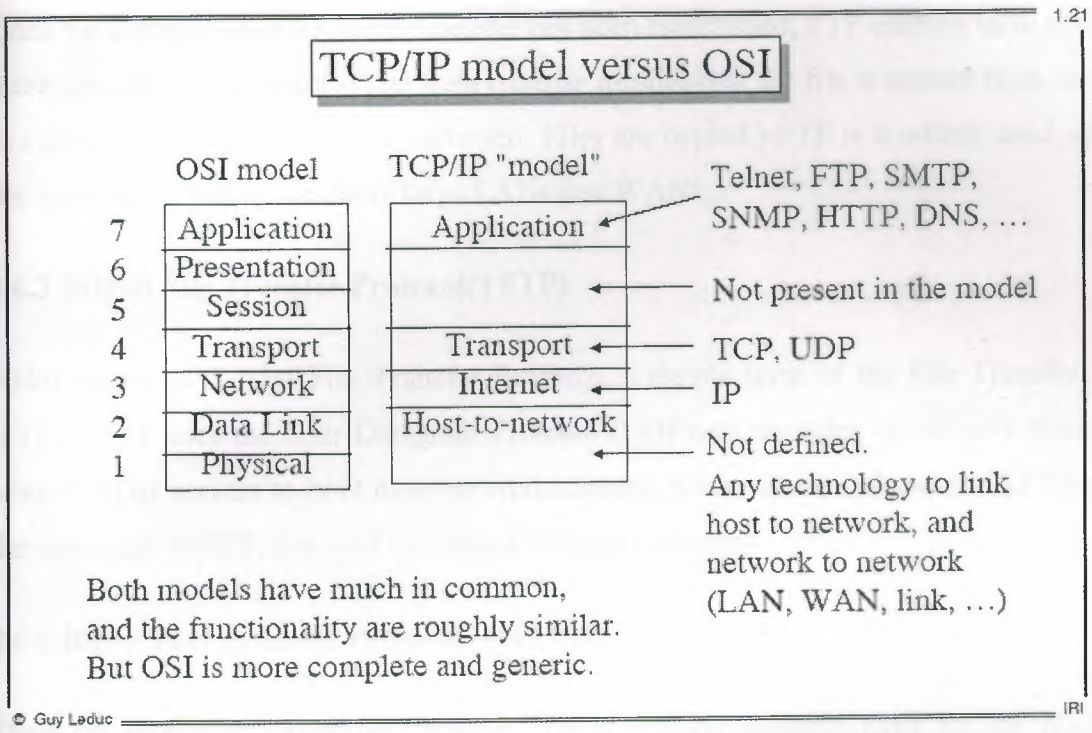
**packet-switched network** A network on which the data in a communication takes several paths.**Packet** A unit of data that is prepared for transmission onto a network.
The illustration below shows how TCP/IP moves data. Notice that there are several routes that the data packets can follow from the source to the destination. Unlike the illustration on the preceding page, the data packets here use a variety of routes-some follow the same path, while others follow different paths. Each packet follows its own route, and data is reassembled at the destination. This is how information moves on a packet-switched network.




### 4.6 An Overview of TCP/IP Components

To understand the roles of the many components of the TCP/IP protocol family, it is useful to know what you can do over a TCP/IP network. Then, once the applications are understood,

the protocols that make it possible are a little easier to comprehend. The following list is not exhaustive but mentions the primary user applications that TCP/IP provides.




### 4.6.1 Telnet

The Telnet program provides a remote login capability. This lets a user on one machine log onto another machine and act as though he or she were directly in front of the second machine. The connection can be anywhere on the local network or on another network anywhere in the world, as long as the user has permission to log onto the remote system.

We can use Telnet when we need to perform actions on a machine across the country. This isn't often done except in a LAN or WAN context, but a few systems accessible through the Internet allow Telnet sessions while users play around with a new application or operating system

### 4.6.2 File Transfer Protocol(FTP)

File Transfer Protocol (FTP) enables a file on one system to be copied to another system. The user doesn't actually log in as a full user to the machine he or she wants to access, as with

Telnet, but instead uses the FTP program to enable access. Again, the correct permissions are necessary to provide access to the files.

Once the connection to a remote machine has been established, FTP enables us to copy one or more files to your machine. (The term *transfer* implies that the file is moved from one system to another but the original is not affected. Files are copied.) FTP is a widely used service on the Internet, as well as on many large LANs and WANs.

### 4.6.3 Trivial File Transfer Protocol(TFTP)

Abbreviation of **T**rivial **F**ile **T**ransfer **P**rotocol, a simple form of the File Transfer Protocol (FTP). TFTP uses the User Datagram Protocol (UDP)and provides no security features. It is often used by servers to boot diskless workstations, X-terminals, and routers. TFTP performs the same task as FTP, but uses a different transport protocol.

### 4.6.4 Hiper Text Transfer Protocol( HTTP)

Short for **H**yper**T**ext **T**ransfer **P**rotocol, the underlying protocol used by the World Wide Web. HTTP defines how messages are formatted and transmitted, and what actions Web servers and browsers should take in response to various commands. For example, when you enter a URL in your browser, this actually sends an HTTP command to the Web server directing it to fetch and transmit the requested Web page.

The other main standard that controls how the World Wide Web works is HTML, which covers how Web pages are formatted and displayed.

HTTP is called a stateless protocol because each command is executed independently, without any knowledge of the commands that came before it. This is the main reason that it is difficult to implement Web sites that react intelligently to user input. This shortcoming of HTTP is being addressed in a number of new technologies, including ActiveX, Java, JavaScript and cookies.

### 4.6.5 Simple Mail Transfer Protocol(SMTP)

Simple Mail Transfer Protocol (SMTP) is used for transferring electronic mail. SMTP is completely transparent to the user. Behind the scenes, SMTP connects to remote machines

and transfers mail messages much like FTP transfers files. Users are almost never aware of SMTP working, and few system administrators have to bother with it. SMTP is a mostly trouble-free protocol and is in very wide use.

### 4.6.6 Kerberos

Kerberos is a widely supported security protocol. Kerberos uses a special application called an authentication server to validate passwords and encryption schemes. Kerberos is one of the more secure encryption systems used in communications and is quite common in UNIX.

### 4.6.7 Domain Name System(DNS)

Domain Name System (DNS) enables a computer with a common name to be converted to a special network address. For example, a PC called Darkstar cannot be accessed by another machine on the same network (or any other connected network) unless some method of checking the local machine name and replacing the name with the machine's hardware address is available. DNS provides a conversion from the common local name to the unique physical address of the device's network connection.

### 4.6.8 Simple Network Management Protocol(SNMP)

Simple Network Management Protocol (SNMP) provides status messages and problem reports across a network to an administrator. SNMP uses User Datagram Protocol (UDP) as a transport mechanism. SNMP employs slightly different terms from TCP/IP, working with managers and agents instead of clients and servers (although they mean essentially the same thing). An agent provides information about a device, whereas a manager communicates across a network with agents.

### 4.6.9 Network File System(NFS)

Network File System (NFS) is a set of protocols developed by Sun Microsystems to enable multiple machines to access each other's directories transparently. They accomplish this by using a distributed file system scheme. NFS systems are common in large corporate environments, especially those that use UNIX workstations.

### 4.6.10 Remote Procedure Call(RPC)

The Remote Procedure Call (RPC) protocol is a set of functions that enable an application to communicate with another machine (the server). It provides for programming functions, return codes, and predefined variables to support distributed computing.

### 4.6.11 Transmission Control Protocol(TCP)

Transmission Control Protocol (the TCP part of TCP/IP) is a communications protocol that provides reliable transfer of data. It is responsible for assembling data passed from higher-layer applications into standard packets and ensuring that the data is transferred correctly.

### 4.6.12 User Datagram Protocol(UDP)

User Datagram Protocol (UDP) is a connectionless-oriented protocol, meaning that it does not provide for the retransmission of datagrams (unlike TCP, which is connection-oriented). UDP is not very reliable, but it does have specialized purposes. If the applications that use UDP have reliability checking built into them, the shortcomings of UDP are overcome.

### 4.6.13 Internet Protocol(IP)

Internet Protocol (IP) is responsible for moving the packets of data assembled by either TCP or UDP across networks. It uses a set of unique addresses for every device on the network to determine routing and destinations.

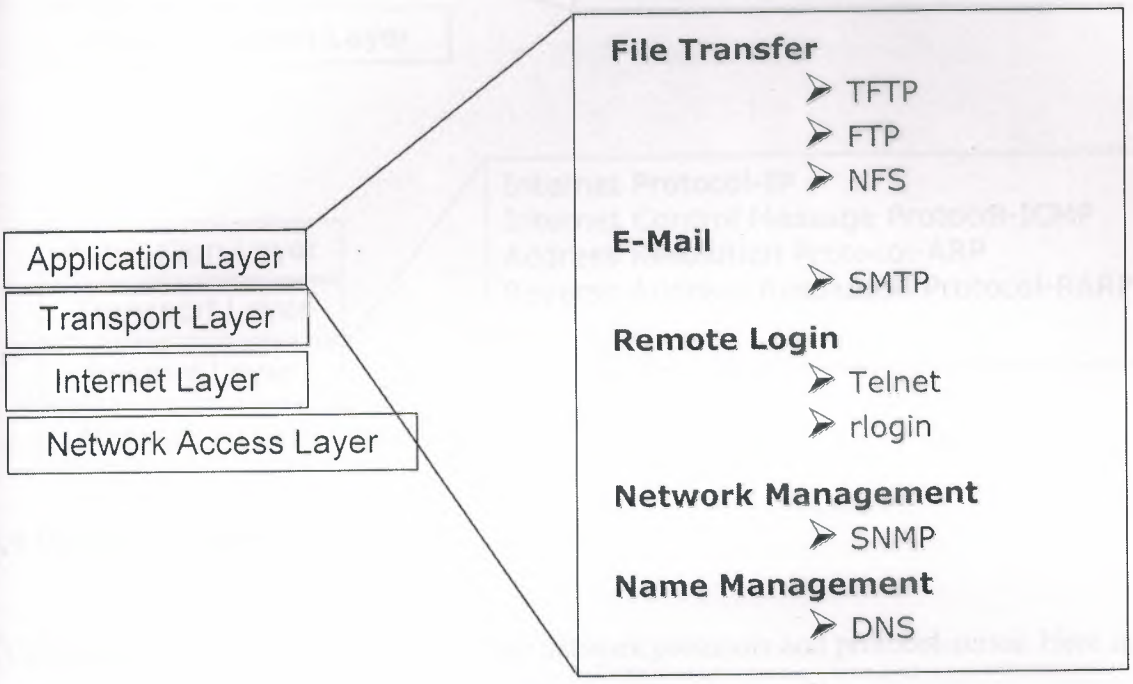
### 4.6.14 Internet Control Message Protocol

Internet Control Message Protocol (ICMP) is responsible for checking and generating messages on the status of devices on a network. It can be used to inform other devices of a failure in one particular machine. ICMP and IP usually work together.

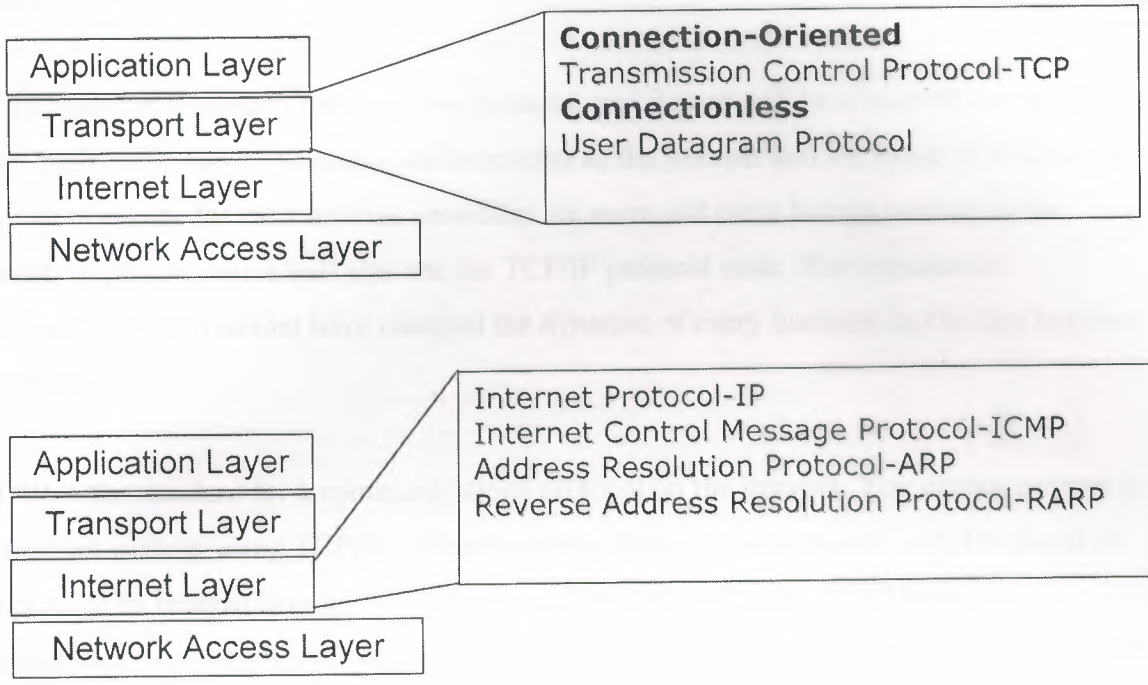
### 4.6.15 X Windows

Serves as a distributed windowing and graphics system used for communication between X terminals and UNIX workstations .

## 4.7 TCP/IP Protocols-1



# 4.8 TCP/IP Protocols-2




## 4.9 Why Use TCP/IP?

TCP/IP offers many advantages over other network protocols and protocol suites. Here is a summary of some of the benefits of using the TCP/IP protocol suite:

• **Widely published, open standard** TCP/IP is not a secret. It is not proprietary or owned by any corporation. Because it is a published protocol with no secrets, any computer engineer is able to improve or enhance the protocol by publishing an RFC.

• **Compatible with different computer systems** TCP/IP enables any system to communicate with any other system. It is like a universal language that would enable people from any country to communicate effectively with people from any other country.

• **Works on different hardware and network configurations** TCP/IP is accepted and can be configured for virtually every network created.

• **Routable protocol** TCP/IP can figure out the path of every piece of data as it moves through the network. Because TCP/IP is a routable protocol, the size of any TCP/IP network is virtually unlimited.

• **Reliable, efficient data delivery** TCP/IP can guarantee that the data is transferred to another host.

• **Single addressing scheme** TCP/IP uses a single and relatively simple addressing scheme. An administrator can transfer knowledge of TCP/IP to any TCP/IP network without relearning the addressing scheme.

The Internet has become a necessity for business, and it soon will be a necessity at home. Many businesses, large and small, are connected to the Internet and are using TCP/IP as the protocol of choice for their internal networks. As more and more homes connect to the Internet, those computers will also use the TCP/IP protocol suite. The commercial implications of the Internet have changed the dynamic of every business model that has ever been taught.

TCP/IP is the standard for a communications protocol on the Internet. You cannot connect to the Internet without using TCP/IP. Whether you build a network at home with two hosts or you manage an **internetwork** at your business with 100,000 hosts, TCP/IP is a communications protocol that will work effectively. TCP/IP can scale to any size environment and is robust enough to connect different types of LANs.

**internetwork** Several smaller networks connected together.

These are a few of the many reasons why network administrators choose to use TCP/IP as the protocol on their networks.

## CHAPTER FIVE:INTERNET PROTOCOL

### 5.1 Background Of Internet Protocols

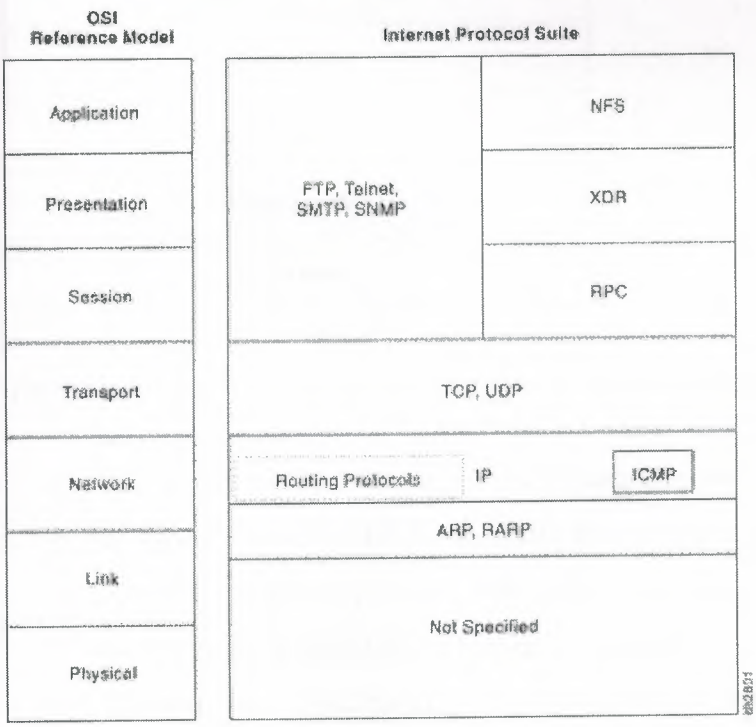
The Internet protocols are the world's most popular open-system (nonproprietary) protocol suite because they can be used to communicate across any set of interconnected networks and are equally well suited for LAN and WAN communications. The Internet protocols consist of a suite of communication protocols, of which the two best known are the Transmission Control Protocol (TCP) and the Internet Protocol (IP). The Internet protocol suite not only includes lower-layer protocols (such as TCP and IP), but it also specifies common applications such as electronic mail, terminal emulation, and file transfer. This chapter provides a broad introduction to specifications that comprise the Internet protocols. Discussions include IP addressing and key upper-layer protocols used in the Internet.

Internet protocols were first developed in the mid-1970s, when the Defense Advanced Research Projects Agency (DARPA) became interested in establishing a packet-switched network that would facilitate communication between dissimilar computer systems at research institutions. With the goal of heterogeneous connectivity in mind, DARPA funded research by Stanford University and Bolt, Beranek, and Newman (BBN). The result of this development effort was the Internet protocol suite, completed in the late 1970s.

TCP/IP later was included with Berkeley Software Distribution (BSD) UNIX and has since become the foundation on which the Internet and the World Wide Web (WWW) are based.

Documentation of the Internet protocols (including new or revised protocols) and policies are specified in technical reports called Request For Comments (RFCs), which are published and then reviewed and analyzed by the Internet community. Protocol refinements are published in the new RFCs. To illustrate the scope of the Internet protocols, Figure 5-1 maps many of the protocols of the Internet protocol suite and their corresponding OSI layers. This chapter addresses the basic elements and operations of these and other key Internet protocols.

**Figure 5-1: Internet protocols span the complete range of OSI model layers.**



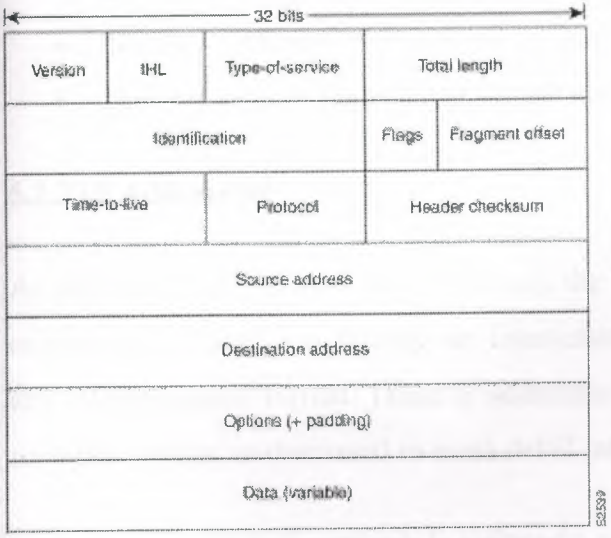
## 5.2 Internet Protocol (IP)

The Internet Protocol (IP) is a network-layer (Layer 3) protocol that contains addressing information and some control information that enables packets to be routed. IP is documented in RFC 791 and is the primary network-layer protocol in the Internet protocol suite. Along with the Transmission Control Protocol (TCP), IP represents the heart of the Internet protocols. IP has two primary responsibilities: providing connectionless, best-effort delivery of datagrams through an internetwork; and providing fragmentation and reassembly of datagrams to support data links with different maximum-transmission unit (MTU) sizes.

### 5.2.1 IP Packet Format

An IP packet contains several types of information, as illustrated in Figure 5-2.

**Figure 5-2: Fourteen fields comprise an IP packet.**




The following discussion describes the IP packet fields illustrated in Figure 5-2:

- *Version*---Indicates the version of IP currently used.
- *IP Header Length* (IHL)---Indicates the datagram header length in 32-bit words.
- *Type-of-Service*---Specifies how an upper-layer protocol would like a current datagram to be handled, and assigns datagrams various levels of importance.
- *Total Length*---Specifies the length, in bytes, of the entire IP packet, including the data and header.
- *Identification*---Contains an integer that identifies the current datagram. This field is used to help piece together datagram fragments.

- *Flags*---Consists of a 3-bit field of which the two low-order (least-significant) bits control fragmentation. The low-order bit specifies whether the packet can be fragmented. The middle bit specifies whether the packet is the last fragment in a series of fragmented packets. The third or high-order bit is not used.
- *Fragment Offset*---Indicates the position of the fragment's data relative to the beginning of the data in the original datagram, which allows the destination IP process to properly reconstruct the original datagram.
- *Time-to-Live*---Maintains a counter that gradually decrements down to zero, at which point the datagram is discarded. This keeps packets from looping endlessly.
- *Protocol*---Indicates which upper-layer protocol receives incoming packets after IP processing is complete.
- *Header Checksum*---Helps ensure IP header integrity.
- *Source Address*---Specifies the sending node.
- *Destination Address*---Specifies the receiving node.
- *Options*---Allows IP to support various options, such as security.
- *Data*---Contains upper-layer information.

### 5.2.2 IP Addressing

As with any other network-layer protocol, the IP addressing scheme is integral to the process of routing IP datagrams through an internetwork. Each IP address has specific components and follows a basic format. These IP addresses can be subdivided and used to create addresses for subnetworks, as discussed in more detail later in this chapter.

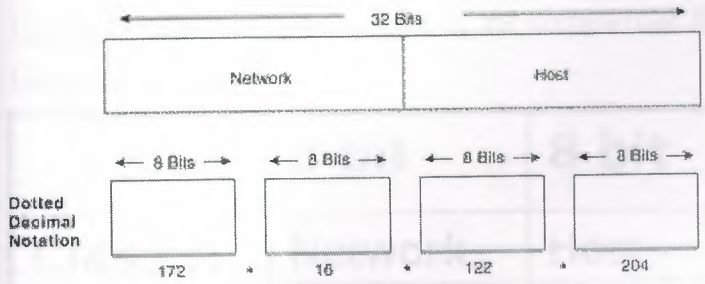
Each host on a TCP/IP network is assigned a unique 32-bit logical address that is divided into two main parts: the network number and the host number. The network number identifies a network and must be assigned by the Internet Network Information Center (InterNIC) if the network is to be part of the Internet. An Internet Service Provider (ISP) can obtain blocks of network addresses from the InterNIC and can itself assign address space as necessary. The host number identifies a host on a network and is assigned by the local network administrator.

### 5.2.3 IP Address Format

The 32-bit IP address is grouped eight bits at a time, separated by dots, and represented in decimal format (known as dotted decimal notation). Each bit in the octet has a binary weight

(128, 64, 32, 16, 8, 4, 2, 1). The minimum value for an octet is 0, and the maximum value for an octet is 255. Figure5-3 illustrates the basic format of an IP address.

**Figure 5-3: An IP address consists of 32 bits, grouped into four octets.**




### 5.2.4 IP Address Classes

IP addressing supports five different address classes: A, B,C, D, and E. Only classes A, B, and C are available for commercial use. The left-most (high-order) bits indicate the network class. Table 5-4 provides reference information about the five IP address classes.

**Table 5-4: Reference Information About the Five IP Address Classes**

| IP Address Class | Format | Purpose | High-Order Bit(s) | Address Range | No. Bits Network/Host | Max. Hosts |
|---|---|---|---|---|---|---|
| A | N.H.H.H[1] | Few large organizations | 0 | 1.0.0.0 to 126.0.0.0 | 7/24 | 16,777, 214[2] ($2^{24}$ - 2) |
| B | N.N.H.H | Medium-size organizations | 1, 0 | 128.1.0.0 to 191.254.0.0 | 14/16 | 65, 543 ($2^{16}$ - 2) |
| C | N.N.N.H | Relatively small organizations | 1, 1, 0 | 192.0.1.0 to 223.255.254.0 | 22/8 | 245 ($2^{8}$ - 2) |
| D | N/A | Multicast groups (RFC 1112) | 1, 1, 1, 0 | 224.0.0.0 to 239.255.255.255 | N/A (not for commercial use) | N/A |
| E | N/A | Experimental | 1, 1, 1, 1 | 240.0.0.0 to 254.255.25 | N/A | N/A |

| | | | | | | |
|---|---|---|---|---|---|---|
| | | | | 5.255 | | |

[1]N = Network number, H = Host number.

[2]One address is reserved for the broadcast address, and one address is reserved for the network.

Figure 5-4 illustrates the format of the commercial IP address classes. (Note the high-order bits in each class.)

| | 8 bit | 8 bit | 8 bit | 8 bit |
|---|---|---|---|---|
| **Class A** | Network | Host | Host | Host |
| **Class B** | Network | Network | Host | Host |
| **Class C** | Network | Network | Network | Host |
| **Class D** | Multicast | | | |
| **Class E** | Research | | | |

**IP CLASS I**

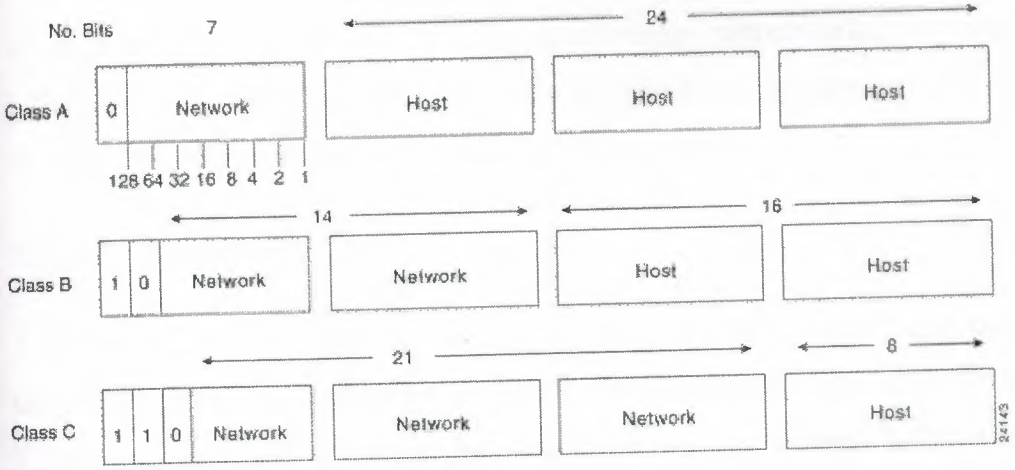
| Class | First Octet | Acceptable Network ID | Number of Network | Total Host number of a Network |
|---|---|---|---|---|
| **A** | 1-126 | 1.0.0.0-126.0.0.0 | $2^7$-2 | $2^{24}$-2 |
| **B** | 128-191 | 128.1.0.0-191.254.0.0 | $2^{14}$-2 | $2^{16}$-2 |
| **C** | 192-223 | 192.0.1.0-223.255.254.0 | $2^{21}$-2 | $2^8$-2 |

IP CLASS II

| CLASS | | FIRST OCTET | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| A | 1-126 | 0 | - | - | - | - | - | - | - |
| B | 128-191 | 1 | 0 | - | - | - | - | - | - |
| C | 192-223 | 1 | 1 | 0 | - | - | - | - | - |
| D | 224-239 | 1 | 1 | 1 | 0 | - | - | - | - |
| E | 240-254 | 1 | 1 | 1 | 1 | 0 | - | - | - |

IP networks can be divided into smaller networks called subnetworks (or subnets). Subnetting provides extra flexibility, makes more efficient use of network address utilization, and contains broadcast traffic because a broadcast will not cross a router. Subnets are under local administration. As such, the outside world sees an organization as a single network, and has no detailed knowledge of the organization's internal structure. A given network address can be broken up into many subnetworks. For example, 172.16.1.0, 172.16.2.0, 172.16.3.0, and 172.16.4.0 are all subnets within network 171.16.0.0. (All 0s in the host portion of an address specifies the entire network.)

**Figure 5-5: IP address formats A, B, and C are available for commercial use.**



The class of address can be determined easily by examining the first octet of the address and mapping that value to a class range in the following table. In an IP address of 172.31.1.2, for example, the first octet is 172. Because 172 falls between 128 and 191, 172.31.1.2 is a Class B address. Figure 5-6 summarizes the range of possible values for the first octet of each address class.

**Figure 5-6: A range of possible values exists for the first octet of each address class.**

| Address Class | First Octet in Decimal | High-Order Bits |
|---|---|---|
| Class A | 1 Ð 126 | 0 |
| Class B | 128 Ð 191 | 10 |
| Class C | 192 Ð 223 | 110 |
| Class D | 224 Ð 239 | 1110 |
| Class E | 240 Ð 254 | 1111 |

24144

### 5.2.5 IP Subnet Addressing

IP networks can be divided into smaller networks called subnetworks (or subnets). Subnetting provides the network administrator with several benefits, including extra flexibility, more efficient use of network addresses, and the capability to contain broadcast traffic (a broadcast will not cross a router).

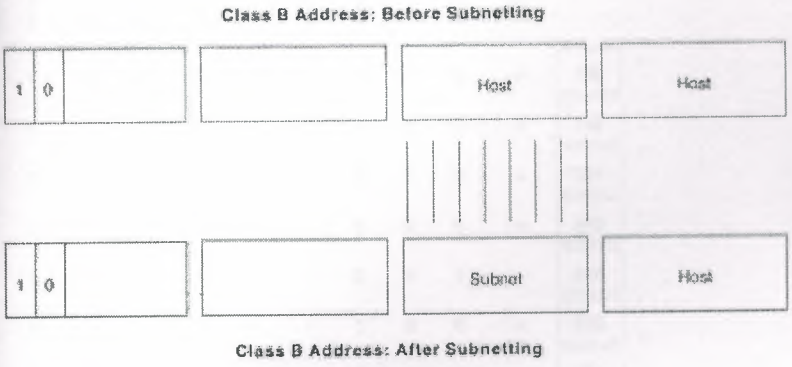
Subnets are under local administration. As such, the outside world sees an organization as a single network and has no detailed knowledge of the organization's internal structure.

A given network address can be broken up into many subnetworks. For example, 172.16.1.0, 172.16.2.0, 172.16.3.0, and 172.16.4.0 are all subnets within network 171.16.0.0. (All 0s in the host portion of an address specifies the entire network.)

### 5.2.6 IP Subnet Mask

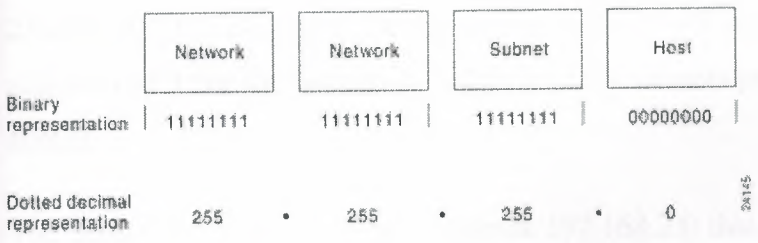
A subnet address is created by "borrowing" bits from the host field and designating them as the subnet field. The number of borrowed bits varies and is specified by the subnet mask. Figure 5-7 shows how bits are borrowed from the host address field to create the subnet address field.

**Figure 5-7: Bits are borrowed from the host address field to create the subnet address field.**



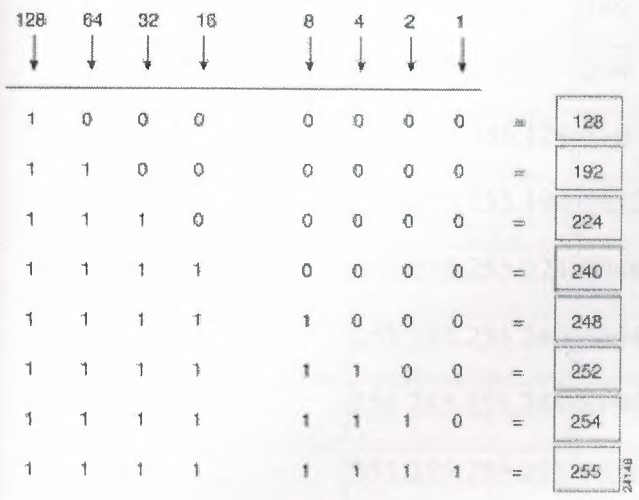

Subnet masks use the same format and representation technique as IP addresses. The subnet mask, however, has binary 1s in all bits specifying the network and subnetwork fields, and binary 0s in all bits specifying the host field. Figure 5-8 illustrates a sample subnet mask.

**Figure 5-8: A sample subnet mask consists of all binary 1s and 0s.**




Subnet mask bits should come from the high-order (left-most) bits of the host field, as Figure 5-9 illustrates. Details of Class B and C subnet mask types follow. Class A addresses are not discussed in this chapter because they generally are subnetted on an 8-bit boundary.

**Figure 5-9: Subnet mask bits come from the high-order bits of the host field.**




Various types of subnet masks exist for Class B and C subnets.

The default subnet mask for a Class B address that has no subnetting is 255.255.0.0, while the subnet mask for a Class B address 171.16.0.0 that specifies eight bits of subnetting is 255.255.255.0. The reason for this is that eight bits of subnetting or $2^8$ - 2 (1 for the network address and 1 for the broadcast address) = 254 subnets possible, with $2^8$ - 2 = 254 hosts per subnet.

The subnet mask for a Class C address 192.168.2.0 that specifies five bits of subnetting is 255.255.255.248.With five bits available for subnetting, $2^5$ - 2 = 30 subnets possible, with $2^3$ - 2 = 6 hosts per subnet.

The reference charts shown in table 30-2 and table 30-3 can be used when planning Class B and C networks to determine the required number of subnets and hosts, and the appropriate subnet mask.

**Table 5-10: Class B Subnetting Reference Chart**

| Number of Bits | Subnet Mask | Number of Subnets | Number of Hosts |
|---|---|---|---|
| 2 | 255.255.192.0 | 2 | 16382 |
| 3 | 255.255.224.0 | 6 | 8190 |
| 4 | 255.255.240.0 | 14 | 4094 |
| 5 | 255.255.248.0 | 30 | 2046 |

| | | | |
|---|---|---|---|
| 6 | 255.255.252.0 | 62 | 1022 |
| 7 | 255.255.254.0 | 126 | 510 |
| 8 | 255.255.255.0 | 254 | 254 |
| 9 | 255.255.255.128 | 510 | 126 |
| 10 | 255.255.255.192 | 1022 | 62 |
| 11 | 255.255.255.224 | 2046 | 30 |
| 12 | 255.255.255.240 | 4094 | 14 |
| 13 | 255.255.255.248 | 8190 | 6 |
| 14 | 255.255.255.252 | 16382 | 2 |

**Table 5-11: Class C Subnetting Reference Chart**

| Number of Bits | Subnet Mask | Number of Subnets | Number of Hosts |
|---|---|---|---|
| 2 | 255.255.255.192 | 2 | 62 |
| 3 | 255.255.255.224 | 6 | 30 |
| 4 | 255.255.255.240 | 14 | 14 |
| 5 | 255.255.255.248 | 30 | 6 |
| 6 | 255.255.255.252 | 62 | 2 |

### 5.2.7 How Subnet Masks are Used to Determine the Network Number

The router performs a set process to determine the network (or more specifically, the subnetwork) address. First, the router extracts the IP destination address from the incoming packet and retrieves the internal subnet mask. It then performs a *logical AND* operation to obtain the network number. This causes the host portion of the IP destination address to be removed, while the destination network number remains. The router then looks up the destination network number and matches it with an outgoing interface. Finally, it forwards the frame to the destination IP address. Specifics regarding the logical AND operation are discussed in the following section.

### Logical AND Operation

Three basic rules govern logically "ANDing" two binary numbers. First, 1 "ANDed" with 1 yields 1. Second, 1 "ANDed" with 0 yields 0. Finally, 0 "ANDed" with 0 yields 0. The truth table provided in table 30-4 illustrates the rules for logical AND operations.

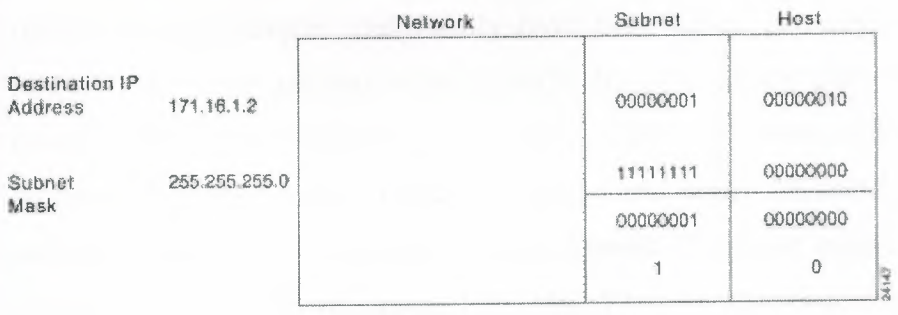
**Table 5-12: Rules for Logical AND Operations**

| Input | Input | Output |
|---|---|---|
| 1 | 1 | 1 |
| 1 | 0 | 0 |
| 0 | 1 | 0 |
| 0 | 0 | 0 |

Two simple guidelines exist for remembering logical AND operations: Logically "ANDing" a 1 with a 1 yields the original value, and logically "ANDing" a 0 with any number yields 0.

Figure 5-13 illustrates that when a logical AND of the destination IP address and the subnet mask is performed, the subnetwork number remains, which the router uses to forward the packet.

**Figure 5-13: Applying a logical AND the destination IP address and the subnet mask produces the subnetwork number.**

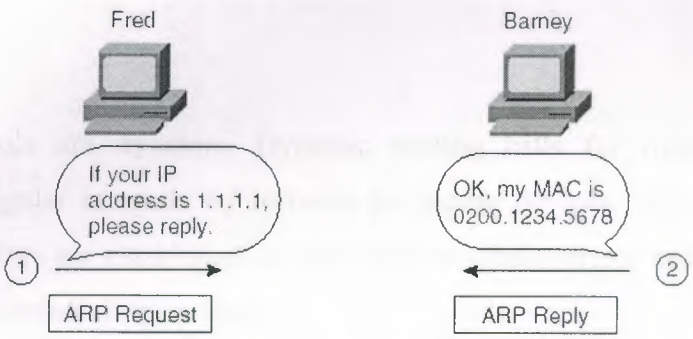



### 5.3 Address Resolution Protocol (ARP)

For two machines on a given network to communicate, they must know the other machine's physical (or MAC) addresses. By broadcasting Address Resolution Protocols (ARPs), a host

can dynamically discover the MAC-layer address corresponding to a particular IP network-layer address.

ARP is a protocol that can **resolve** an IP address to a hardware address. After the hardware address is resolved, ARP maintains that information for a short time. Because the host wants to communicate with another host, but only has the IP address, ARP will ask, "Hey, what is your hardware address?" and wait for an answer.




The first place that ARP looks to resolve an IP address to a hardware address is in **ARP cache**. ARP cache is an area in random access memory (RAM) where ARP keeps the IP and hardware addresses that have been resolved. If ARP can find the IP and hardware addresses in ARP cache, the packet is addressed to the hardware address with no further resolution. If the IP address is not in ARP cache, ARP will initiate an **ARP request** broadcast.

### 5.4 Internet Routing

Internet routing devices traditionally have been called gateways. In today's terminology, however, the term gateway refers specifically to a device that performs application-layer protocol translation between devices. Interior gateways refer to devices that perform these protocol functions between machines or networks under the same administrative control or authority, such as a corporation's internal network. These are known as autonomous systems. Exterior gateways perform protocol functions between independent networks.

Routers within the Internet are organized hierarchically. Routers used for information exchange within autonomous systems are called interior routers, which use a variety of Interior Gateway Protocols (IGPs) to accomplish this purpose. The Routing Information Protocol (RIP) is an example of an IGP.

Routers that move information between autonomous systems are called exterior routers. These routers use an exterior gateway protocol to exchange information between autonomous systems. The Border Gateway Protocol (BGP) is an example of an exterior gateway protocol.

---

**Note** Specific routing protocols, including BGP and RIP, are addressed in individual chapters presented in Part 6 later in this book.

---

### 5.5 IP Routing

IP routing protocols are dynamic. Dynamic routing calls for routes to be calculated automatically at regular intervals by software in routing devices. This contrasts with static routing, where routers are established by the network administrator and do not change until the network administrator changes them.

An IP routing table, which consists of destination address/next hop pairs, is used to enable dynamic routing. An entry in this table, for example, would be interpreted as follows: to get to network 172.31.0.0, send the packet out Ethernet interface 0 (E0).

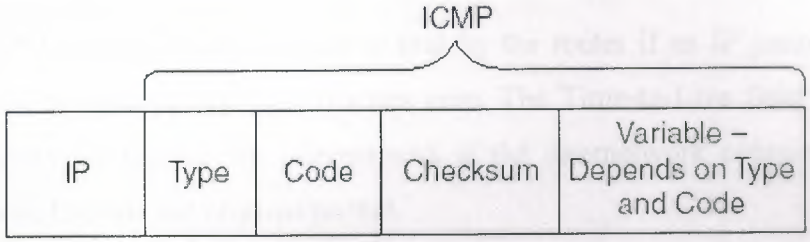
IP routing specifies that IP datagrams travel through internetworks one hop at a time. The entire route is not known at the onset of the journey, however. Instead, at each stop, the next destination is calculated by matching the destination address within the datagram with an entry in the current node's routing table.

Each node's involvement in the routing process is limited to forwarding packets based on internal information. The nodes do not monitor whether the packets get to their final destination, nor does IP provide for error reporting back to the source when routing anomalies occur. This task is left to another Internet protocol, the Internet Control-Message Protocol (ICMP), which is discussed in the following section.

### 5.6 Internet Control Message Protocol (ICMP)

The Internet Control Message Protocol (ICMP) is a network-layer Internet protocol that provides message packets to report errors other information relevant to IP packet processing back to the source. ICMP is documented in RFC 792.

**Format of ICMP**:




### 5.6.1 ICMP Messages

ICMPs generate several kinds of useful messages, including Destination Unreachable, Echo Request and Reply, Redirect, Time Exceeded, and Router Advertisement and Router Solicitation. If an ICMP message cannot be delivered, no second one is generated. This is to avoid an endless flood of ICMP messages.

When an ICMP destination-unreachable message is sent by a router, it means that the router is unable to send the package to its final destination. The router then discards the original packet. Two reasons exist for why a destination might be unreachable. Most commonly, the source host has specified a nonexistent address. Less frequently, the router does not have a route to the destination.

Destination-unreachable messages include four basic types: network unreachable, host unreachable, protocol unreachable, and port unreachable. Network-unreachable messages usually mean that a failure has occurred in the routing or addressing of a packet. *Host*-unreachable messages usually indicates delivery failure, such as a wrong subnet mask. Protocol-unreachable messages generally mean that the destination does not support the upper-layer protocol specified in the packet. Port-unreachable messages imply that the TCP socket or port is not available.

An ICMP echo-request message, which is generated by the ping command, is sent by any host to test node reachability across an internetwork. The ICMP echo-reply message indicates that the node can be successfully reached.

An ICMP Redirect message is sent by the router to the source host to stimulate more efficient routing. The router still forwards the original packet to the destination. ICMP redirects allow host routing tables to remain small because it is necessary to know the address of only one

router, even if that router does not provide the best path. Even after receiving an ICMP Redirect message, some devices might continue using the less-efficient route.

An ICMP Time-exceeded message is sent by the router if an IP packet's Time-to-Live field (expressed in hops or seconds) reaches zero. The Time-to-Live field prevents packets from continuously circulating the internetwork if the internetwork contains a routing loop. The router then discards the original packet.

**ICMP provides a number of helpful messages including the following:**

**Destination Unreachable** - The ICMP destination unreachable message is sent by a router if it is unable to deliver a packet to the ultimate destination. The router discards the original packet. Destinations might be unreachable for these reasons:

The source host specified a nonexistent address.

The router does not have a route to the destination (less frequent).

Destination unreachable messages include the following:

Network unreachable -- This message usually implies routing or addressing failures.

Host unreachable -- This message usually implies delivery failures such as a wrong subnet mask.

Protocol unreachable -- This message usually implies that the destination does not support the upper-layer protocol specified in the packet.

Port unreachable -- This message usually implies that the Transmission Control Protocol (TCP) port (socket) is not available.

**Echo Request and Reply** - The ICMP echo request message is sent by any host to test node reachability across an internetwork. It is generated by the ping command. The ICMP echo reply message indicates that the node can be successfully reached.

**Redirect** - An ICMP redirect message is sent by the router to the source host to stimulate more efficient routing. The router still forwards the original packet to the destination. ICMP redirects allow host routing tables to remain small because knowing the address of only one router is required (even if that router does not provide the best path). Even after receiving an ICMP redirect message, some devices might continue using the less efficient route.

**Time Exceeded** - An ICMP time-exceeded message is sent by the router if an IP packet's Time-to-Live field (expressed in hops or seconds) reaches zero. The Time-to-Live field

prevents packets from continuously circulating the internetwork if the internetwork contains a routing loop. The router discards the original packet.

**Router Advertisement and Router Solicitation** - The ICMP Router Discovery Protocol (IDRP) uses router advertisement and router solicitation messages to discover the addresses of routers on directly attached subnets. IDRP works as follows:

1.Each router periodically multicasts router advertisement messages from each of its interfaces.

2.Hosts discover addresses of routers on directly attached subnets by listening for these messages.

3.Hosts can use router solicitation messages to request immediate advertisements, rather than waiting for unsolicited messages.

IRDP offers several advantages over other methods of discovering addresses of neighboring routers. Primarily, it does not require hosts to recognize routing protocols, nor does it require manual configuration by an administrator. Router advertisement messages allow hosts to discover the existence of neighboring routers, but not which router is best to reach a particular destination. If a host uses a poor first-hop router to reach a particular destination, it receives a redirect message identifying a better choice.

Undeliverable ICMP messages (for whatever reason) do not generate a second ICMP message. Doing so could create an endless flood of ICMP messages.

### 5.6.2 ICMP Router-Discovery Protocol (IDRP)

IDRP uses Router-Advertisement and Router-Solicitation messages to discover the addresses of routers on directly attached subnets. Each router periodically multicasts Router-Advertisement messages from each of its interfaces. Hosts then discover addresses of routers on directly attached subnets by listening for these messages. Hosts can use Router-Solicitation messages to request immediate advertisements rather than waiting for unsolicited messages.

IRDP offers several advantages over other methods of discovering addresses of neighboring routers. Primarily, it does not require hosts to recognize routing protocols, nor does it require manual configuration by an administrator.

Router-Advertisement messages enable hosts to discover the existence of neighboring routers, but not which router is best to reach a particular destination. If a host uses a poor first-hop

router to reach a particular destination, it receives a Redirect message identifying a better choice.

## CHAPTER SIX:TRANSMISSION CONTROL PROTOCOL

### 6.1 Transmission Control Protocol (TCP)

The TCP provides reliable transmission of data in an IP environment. TCP corresponds to the transport layer (Layer 4) of the OSI reference model. Among the services TCP provides are stream data transfer, reliability, efficient flow control, full-duplex operation, and multiplexing.

With stream data transfer, TCP delivers an unstructured stream of bytes identified by sequence numbers. This service benefits applications because they do not have to chop data into blocks before handing it off to TCP. Instead, TCP groups bytes into segments and passes them to IP for delivery.

TCP offers reliability by providing connection-oriented, end-to-end reliable packet delivery through an internetwork. It does this by sequencing bytes with a forwarding acknowledgment number that indicates to the destination the next byte the source expects to receive. Bytes not acknowledged within a specified time period are retransmitted. The reliability mechanism of TCP allows devices to deal with lost, delayed, duplicate, or misread packets. A time-out mechanism allows devices to detect lost packets and request retransmission.

TCP offers efficient flow control, which means that, when sending acknowledgments back to the source, the receiving TCP process indicates the highest sequence number it can receive without overflowing its internal buffers.

Full-duplex operation means that TCP processes can both send and receive at the same time.

Finally, TCP's multiplexing means that numerous simultaneous upper-layer conversations can be multiplexed over a single connection.

### 6.2 TCP Connection Establishment

To use reliable transport services, TCP hosts must establish a connection-oriented session with one another. Connection establishment is performed by using a "three-way handshake" mechanism.

A three-way handshake synchronizes both ends of a connection by allowing both sides to agree upon initial sequence numbers. This mechanism also guarantees that both sides are ready to transmit data and know that the other side is ready to transmit as well. This is necessary so that packets are not transmitted or retransmitted during session establishment or after session termination.

Each host randomly chooses a sequence number used to track bytes within the stream it is sending and receiving. Then, the three-way handshake proceeds in the following manner:

The first host (Host A) initiates a connection by sending a packet with the initial sequence number (X) and SYN bit set to indicate a connection request. The second host (Host B) receives the SYN, records the sequence number X, and replies by acknowledging the SYN (with an ACK = X + 1). Host B includes its own initial sequence number (SEQ = Y). An ACK = 20 means the host has received bytes 0 through 19 and expects byte 20 next. This technique is called forward acknowledgment. Host A then acknowledges all bytes Host B sent with a forward acknowledgment indicating the next byte Host A expects to receive (ACK = Y + 1). Data transfer then can begin.

### 6.2.1 Positive Acknowledgment and Retransmission (PAR)

A simple transport protocol might implement a reliability-and-flow-control technique where the source sends one packet, starts a timer, and waits for an acknowledgment before sending a new packet. If the acknowledgment is not received before the timer expires, the source retransmits the packet. Such a technique is called positive acknowledgment and retransmission (PAR).

By assigning each packet a sequence number, PAR enables hosts to track lost or duplicate packets caused by network delays that result in premature retransmission. The sequence numbers are sent back in the acknowledgments so that the acknowledgments can be tracked.

PAR is an inefficient use of bandwidth, however, because a host must wait for an acknowledgment before sending a new packet, and only one packet can be sent at a time.

### 6.2.2 TCP Sliding Window

*A* TCP sliding window provides more efficient use of network bandwidth than PAR because it enables hosts to send multiple bytes or packets before waiting for an acknowledgment.

In TCP, the receiver specifies the current window size in every packet. Because TCP provides a byte-stream connection, window sizes are expressed in bytes. This means that a window is the number of data bytes that the sender is allowed to send before waiting for an acknowledgment. Initial window sizes are indicated at connection setup, but might vary throughout the data transfer to provide flow control. A window size of zero, for instance, means "Send no data."

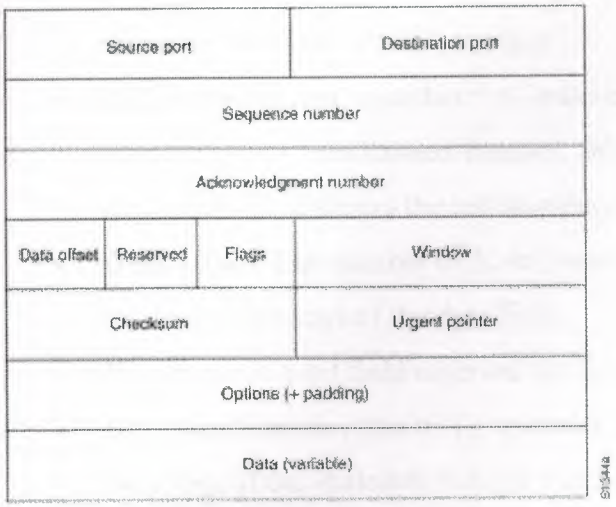
In a TCP sliding-window operation, for example, the sender might have a sequence of bytes to send (numbered 1 to 10) to a receiver who has a window size of five. The sender then would place a window around the first five bytes and transmit them together. It would then wait for an acknowledgment.

The receiver would respond with an ACK = 6, indicating that it has received bytes 1 to 5 and is expecting byte 6 next. In the same packet, the receiver would indicate that its window size is 5. The sender then would move the sliding window five bytes to the right and transmit bytes 6 to 10. The receiver would respond with an ACK = 11, indicating that it is expecting sequenced byte 11 next. In this packet, the receiver might indicate that its window size is 0 (because, for example, its internal buffers are full). At this point, the sender cannot send any more bytes until the receiver sends another packet with a window size greater than 0.

## 6.3 TCP Packet Format

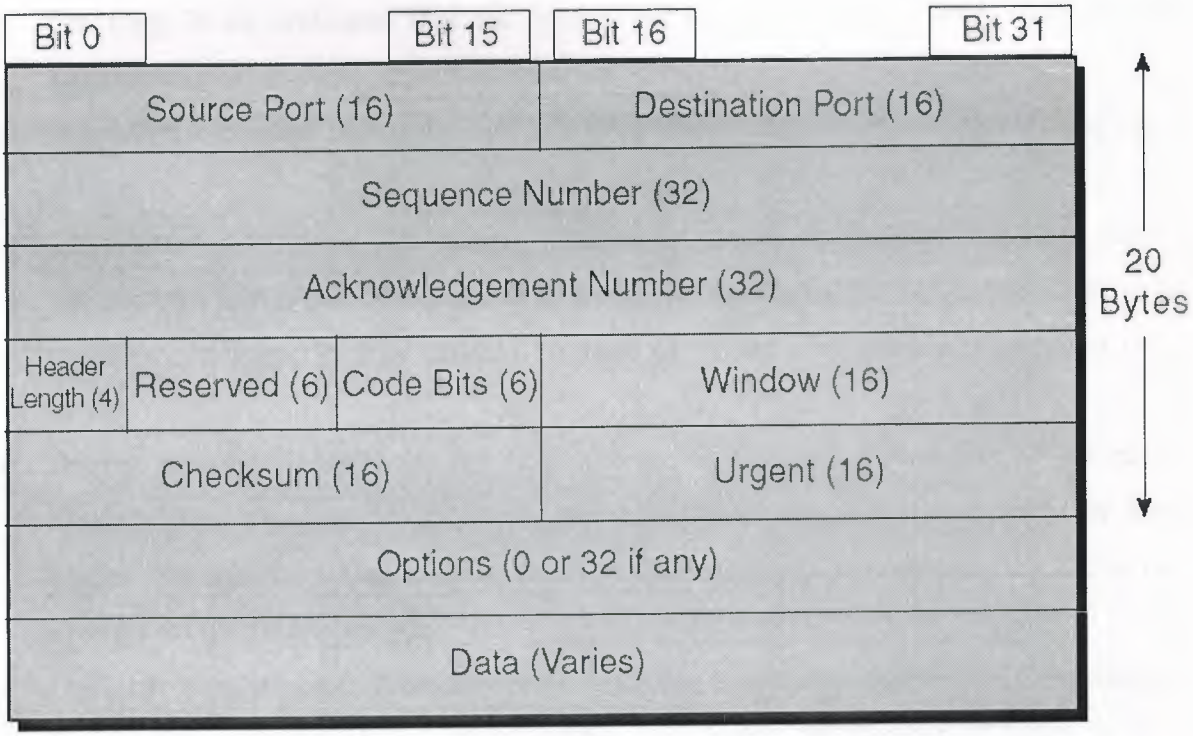
Figure 6-1 illustrates the fields and overall format of a TCP packet.

**Figure 6-1: Twelve fields comprise a TCP packet.**




## 6.4 TCP Packet Field Descriptions

**Figure 6-2:TCP Segment-1**




The different fields are as follows:

- **Source port:** A 16-bit field that identifies the local TCP user (usually an upper-layer application program).
- **Destination port:** A 16-bit field that identifies the remote machine's TCP user.

- **Sequence number:** A number indicating the current block's position in the overall message. This number is also used between two TCP implementations to provide the initial send sequence (ISS) number.
- **Acknowledgment number:** A number that indicates the next sequence number expected. In a backhanded manner, this also shows the sequence number of the last data received; it shows the last sequence number received plus 1.
- **Data offset:** The number of 32-bit words that are in the TCP header. This field is used to identify the start of the data field.
- **Reserved:** A 6-bit field reserved for future use. The six bits must be set to 0.
- **Urg flag:** If on (a value of 1), indicates that the urgent pointer field is significant.
- **Ack flag:** If on, indicates that the Acknowledgment field is significant.
- **Psh flag:** If on, indicates that the push function is to be performed.
- **Rst flag:** If on, indicates that the connection is to be reset.
- **Syn flag:** If on, indicates that the sequence numbers are to be synchronized. This flag is used when a connection is being established.
- **Fin flag:** If on, indicates that the sender has no more data to send. This is the equivalent of an end-of-transmission marker.
- **Window:** A number indicating how many blocks of data the receiving machine can accept.
- **Checksum:** Calculated by taking the 16-bit one's complement of the one's complement sum of the 16-bit words in the header (including pseudo-header) and text together. (A rather lengthy process required to fit the checksum properly into the header.)
- **Urgent pointer:** Used if the urg flag was set; it indicates the portion of the data message that is urgent by specifying the offset from the sequence number in the header. No specific action is taken by TCP with respect to urgent data; the action is determined by the application.
- **Options:** Similar to the IP header option field, this is used for specifying TCP options. Each option consists of an option number (one byte), the number of bytes in the option, and the option values. Only three options are currently defined for TCP:

  0 End of option list

  1 No operation

  2 Maximum segment size
- **Padding:** Filled to ensure that the header is a 32-bit multiple.

## 6.5 User Datagram Protocol (UDP)

The User Datagram Protocol (UDP) is a connectionless transport-layer protocol (Layer 4) that belongs to the Internet protocol family. UDP is basically an interface between IP and upper-layer processes. UDP protocol ports distinguish multiple applications running on a single device from one another.

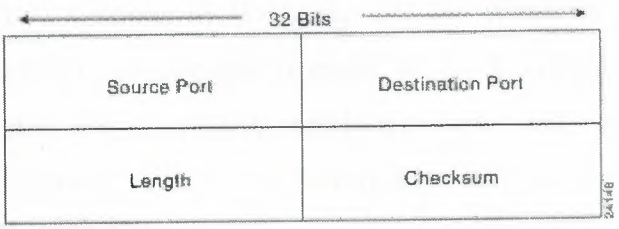
Unlike the TCP, UDP adds no reliability, flow-control, or error-recovery functions to IP. Because of UDP's simplicity, UDP headers contain fewer bytes and consume less network overhead than TCP.

UDP is useful in situations where the reliability mechanisms of TCP are not necessary, such as in cases where a higher-layer protocol might provide error and flow control.

UDP is the transport protocol for several well-known application-layer protocols, including Network File System (NFS), Simple Network Management Protocol (SNMP), Domain Name System (DNS), and Trivial File Transfer Protocol (TFTP).

The UDP packet format contains four fields, as shown in Figure 6-3. These include source and destination ports, length, and checksum fields.

**Figure 6-3: A UDP packet consists of four fields.**




Source and destination ports contain the 16-bit UDP protocol port numbers used to demultiplex datagrams for receiving application-layer processes. A length field specifies the length of the UDP header and data. Checksum provides an (optional) integrity check on the UDP header and data.

## CHAPTER SEVEN:IP VERSIONS

### 7.1 IP versions

This chapter describes the Internet Protocol (IP). This chapter covers IPv4 and IPv6. Chapter 4 will cover the most common protocols .

IPv4 is commonly known as the network protocol that the Internet uses. IP is widely available on most computer operating systems and can be used on most LANs, such as a small network in your office, and on WANs, such as the Internet. With the explosion in the number of computers on the Internet, the limitations of IPv4 are becoming apparent, and as a result, the next generation IP was developed, which is known as IPv6.

In this chapter, we will discuss the background, addressing scheme, name resolution, and Winsock specifics for both IPv4 and IPv6. Then, we'll discuss how to write applications that seamlessly operate over either version of IP.

### 7.2 Winsock

For some Windows and Windows 95 users, Winsock is the easiest method to get into TCP/IP because it is available from many public domain, BBS, and online service sites. There are several versions of Winsock, some of which are public domain or shareware. We will look at two versions of Winsock, one for Windows 3.X and another for Windows 95. We have chosen the popular Trumpet Winsock implementations for both operating systems because they are shareware, readily available, and well supported. Winsock is short for Windows Sockets, originally developed by Microsoft. Released in 1993, Windows Sockets is an interface for network programming in the Windows environment. Microsoft has published the specifications for Windows Sockets, hence making it an open application programming interface (API). The Winsock API (called WSA) is a library of function calls, data structures, and programming procedures that provide this standardized interface for applications. The second release of Winsock, called Winsock version 2, was released in mid 1995.

### 7.3 In Winsock Configuring the TCP/IP Packet Driver

Trumpet Winsock relies on a program called WINPKT to provide TCP/IP packet capabilities under Windows. After you create a program group for Winsock, you need to set up the packet driver information in the network files.

You will need a packet driver for your system, which is not included with most Trumpet Winsock distributions. In many cases, the network card vendor includes a disk with a packet driver on it. If not, one of the best sources for a packet driver is the Crynwr Packet Driver collection, a library of different packet drivers available from many online, BBS, FTP, and WWW sites. The packet driver specifications are added to your network startup batch file, usually AUTOEXEC.BAT for DOS-based systems.

The process for configuring Trumpet Winsock for LAN operation is quite simple. Set the IRQ and I/O address of the packet driver and add the packet driver to your system.

### 7.4 IPv4

IPv4 was developed by the U.S. Department of Defense's Advanced Research Project Agency (ARPA), which built an experimental packet switching network in the 1960s. The initial network protocols were cumbersome, which led to the development of a better protocol in the mid 1970s. This research eventually led to IPv4 as well as TCP.

#### 7.4.1 Addressing

In IPv4, computers are assigned an address that is represented as a 32-bit number, formally known as an IPv4 address. IPv4 addresses are typically represented in a dotted decimal format in which each octet (8 bits) of the address is converted to a decimal number and separated by a period ("dots").

IPv4 addresses are divided into classes that describe the portion of the address assigned to the network and the portion assigned to endpoints. Table 7-1 lists the different classes.

**Table 7-1** *IPv4 Address Classes*

| Class | Network Portion | First Number | Number of Endpoints |
|---|---|---|---|
| A | 8 bits | 0–127 | 16,777,216 |

**Table 7-1** *IPv4 Address Classes*

| Class | Network Portion | First Number | Number of Endpoints |
|---|---|---|---|
| B | 16 bits | 128–191 | 65,536 |
| C | 24 bits | 192–223 | 256 |
| D | N/A | 224–239 | N/A |
| E | N/A | 240–255 | N/A |

When specifying an IP address, the number of bits indicating the network portion can be appended to the dotted decimal address after a back slash (/). For example, the address 172.31.28.120/16 indicates that the first 16 bits make up the network portion of the address. This is equivalent to a subnet mask of 255.255.0.0.

The last two entries in Table 7-1 are special classes of IPv4 addresses. Class D addresses are reserved for IPv4 multicasting and class E addresses are experimental. Also, there are several blocks of addresses that have been reserved for private use and cannot be used by a system on the Internet. They are the following:

- 10.0.0.0–10.255.255.255 (10.0.0.0/8)
- 172.16.0.0–172.31.255.255 (172.16.0.0/12)
- 192.168.0.0–192.168.255.255 (192.168.0.0/16)

Finally, there is the loopback address (127.0.0.1), which is a special address that refers to the local computer.

To list the IPv4 addresses assigned to the local interfaces, the IPCONFIG.EXE command can be used to list each network adapter and the IPv4 address(es) assigned to it. If an application needs to programmatically obtain a list of its IPv4 addresses, it can call WSAIoctl with the SIO_ADDRESS_LIST_QUERY command, which is covered in Chapter 7. In addition, the IP Helper APIs provide this function and are described in Chapter 16.

We've discussed the breakdown of the IPv4 address space, and from within these different address classes there are three types of IPv4 addresses: unicast, multicast, and broadcast. Each address type will be covered in the next sections.

### Unicast

Unicast addresses are those addresses that are assigned to an individual computer interface. Only one interface may be assigned that address. If another computer is configured with the same address on the network, then that is an error that will result in data being delivered incorrectly. Classes A, B, and C comprise the unicast address space for IPv4.

Typically, an interface on a host is assigned an IPv4 (unicast) address either statically or by a configuration protocol like Dynamic Host Configuration Protocol (DHCP). If a DHCP server cannot be reached, the system automatically assigns an address in the range of 169.254.0.0/16 using Automatic Private IP Addressing (APIPA).

To prevent having to memorize numeric IP addresses, an IPv4 address can be associated to the host computer name by using the Domain Name System (DNS). Later, we will discuss how to resolve the host name to its IPv4 address (and its IPv6 address as well).

### Multicast

Multicast addresses are not assigned to a specific interface. Instead, multiple computers may "join" a multicast group listening on a particular multicast address. Everyone joined to that group will receive any data destined to that address. Multicast addresses are class D addresses. One of the greatest benefits to multicasting is the capability to deliver multicast data to only those machines that are interested in that data. IP multicasting is discussed in detail in Chapter 9.

### Broadcast

IPv4 supports broadcasting data. This means that data sent to the limited broadcast address, 255.255.255.255, will be received and processed by every machine on the local network. This is generally considered a bad practice because even those computers that are not interested in the broadcast data must process the packet.

If applications require broadcasting, it is better to use subnet directed broadcasts. This is still broadcasting data, but as the name implies it is directed to machines on a specific subnet only. For example, a datagram sent to 172.31.28.255 will be received by every machine on only that same subnet.

### 7.4.2 IPv4 Management Protocols

The IPv4 protocol relies on several other protocols to function. The three support protocols we are most interested in is the Address Resolution Protocol (ARP), the Internet Control Message Protocol (ICMP), and the Internet Group Management Protocol (IGMP).

ARP is used to resolve the 32-bit IPv4 address into a physical or hardware address so the IPv4 packet can be wrapped in the appropriate media frame (such as an Ethernet frame). A host must resolve the next-hop IPv4 address to its corresponding hardware address before sending data on the wire. If the destination address is on the local network, the ARP request is made for the destination's physical address. If one or more routers separate the source from the destination, an ARP request is made for the default gateway and the packet is forwarded to it. The IP Helper API contains some ARP routines and is described in Chapter 16.

ICMP is designed to send status and error messages between IPv4 hosts. The types of messages include echo requests and replies, destination unreachable, and time exceeded. ICMP is also used to discover nearby routers. Chapter 11 will go into more detail on ICMP and will illustrate how to send your own ICMP messages.

IGMP is used to manage multicast group membership. When applications on a host join multicast group, the host sends out IGMP membership reports, which inform routers on the network segment which multicast groups data is to be received on. Routers need this information to forward multicast packets destined to these multicast groups to network segments only when there are receivers interested in it.

### 7.4.3 Addressing IPv4 from Winsock

In Winsock, applications specify IPv4 addresses and service port information through the SOCKADDR_IN structure, which is defined as

```
struct sockaddr_in
{
  short        sin_family;
  u_short      sin_port;
  struct in_addr  sin_addr;
  char         sin_zero[8];
```

};

The sin_family field must be set to AF_INET, which tells Winsock you are using the IP address family. The sin_port field defines which TCP or UDP communication port will be used to identify a server service. Note that the port number does not actually apply to the IPv4 protocol but is a property of the transport layer protocol(s) encapsulated within an IPv4 header, such as TCP or UDP.

Applications should be particularly careful in choosing a port because some of the available port numbers are reserved for well-known services, such as FTP and HTTP. The ports that well-known services use are controlled and assigned by the Internet Assigned Numbers Authority (IANA) and are listed on its Web page at http://www.iana.org/assignments/port-numbers. Essentially, the port numbers are divided into the following three ranges: well-known, registered, and dynamic and/or private ports.

- 0–1023 are controlled by IANA and are reserved for well-known services.
- 1024–49151 are registered ports listed by IANA and can be used by ordinary user processes or programs executed by ordinary users.
- 49152–65535 are dynamic and/or private ports.

Ordinary user applications should choose the registered ports in the range 1024–49151 to avoid the possibility of using a port already in use by another application or a system service. Ports in the range 49152–65535 can also be used freely because no services are registered on these ports with IANA. If, when using the bind API function, your application binds to a port that is already in use by another application on your host, the system will return the Winsock error WSAEADDRINUSE. Also, it is valid for clients to send or connect without explicitly binding to a local address and port. In this case, the system will implicitly bind the socket to a local port from the range of 1024 to 5000. This is the same behavior that occurs if an application explicitly binds the socket but specifies a local port of zero.

The sin_addr field of the SOCKADDR_IN structure is used for storing an IPv4 address as a four-byte, network-byte-ordered quantity, which is an unsigned long integer data type. Depending on how this field is used, it can represent a local or a remote IP address. IP addresses are normally specified in Internet standard dotted notation as "a.b.c.d." Each letter represents a number for each byte and is assigned, from left to right, to the four bytes of the

unsigned long integer. The final field, sin_zero, functions only as padding to make the SOCKADDR_IN structure the same size as the SOCKADDR structure.

All fields of this and every other socket address structure need to be in network byte order. However, if applications use the name resolution and assignment APIs discussed later in this chapter, the necessary conversion is automatically performed. It is only when an application explicitly assigns or retrieves values from the structure members that the byte order conversion is required. Byte ordering was described in Chapter 1.

### 7.5 IPv6

With the explosion in the number of computers on the Internet, the limitations of IPv4 are becoming apparent. First and foremost, the number of available IPv4 addresses is being exhausted. This has led to the use of network address translators (NATs), which map multiple private addresses to a single public IP addresses. NATs are useful for client-server applications but can be problematic when connecting two organizations that use the private address space. Also, NATs must sometimes be aware of the underlying protocols to perform the appropriate address translation.

Second, IPv4 addressing is not entirely hierarchical, which means that the Internet backbone routers must maintain vast routing tables to deliver IPv4 packets correctly to any location on the Internet.

Another incentive for developing IPv6 is to provide simpler configuration. With IPv4, addresses must be assigned statically or via a configuration protocol such as DHCP. Ideally, hosts would not have to rely upon the administration of a DHCP infrastructure. Instead, they will be able to auto configure themselves based on the network segment on which they are located.

A developer-release version of IPv6 is provided with Windows XP. For Windows 2000, a technology preview IPv6 protocol is available for download from http://www.microsoft.com/ipv6. For Windows NT 4.0, a Microsoft Research IPv6 protocol may also be obtained from http://www.microsoft.com/ipv6.

In this section, we will cover the different types of IPv6 addresses, the support protocols that IPv6 uses, and how IPv6 addresses are handled from Winsock. Although we will discuss

addressing and name resolution, we will not cover all aspects of IPv6, such as routing or setting up an IPv6 network. For more information, consult the Windows XP online help or the book Understanding IPv6, by Joseph Davies (Microsoft Press, 2002).

### 7.5.1 Addressing

The most noticeable difference between IPv4 and IPv6 addresses is that an IPv6 address is 128 bits, which is four times larger than an IPv4 address. One reason for such a large address space is to subdivide the available addresses into a hierarchy of routing domains that reflect the Internet's topology. Table 3-2 lists a portion of how the address space is allocated and lists the address prefix for each portion. The address prefix denotes the high order bits of an IPv6 address. IPv6 addressing is described in RFC 2373.

**Table 7-2** *IPv6 Address Allocation*

| Allocation | Address Prefix | Fraction of Address Space |
|---|---|---|
| Reserved | 0000 0000 | 1/256 |
| Reserved for NSAP allocation | 0000 001 | 1/128 |
| Aggregatable global unicast addresses | 001 | 1/8 |
| Link-local unicast addresses | 1111 1110 10 | 1/1024 |
| Site-local unicast addresses | 1111 1110 11 | 1/1024 |
| Multicast addresses | 1111 1111 | 1/256 |

An IPv6 address is typically expressed in 16-bit chunks displayed as hexadecimal numbers separated by colons. The following is an example of an IPv6 address:

21DA:00D3:0000:2F3B:02AA:00FF:FE28:9C5A

Leading zeroes within each 16-bit block may be removed, as seen here:

21DA:D3:0:2F3B:2AA:FF:FE28:9C5A

Many IPv6 addresses contain long sequences of zeroes, which may be compressed by substituting two colons for the block of zeros. For example, the following address:

FE80:0:0:0:12:0:34:56

can be compressed to:

FE80::12:0:34:56

Note that only a single contiguous sequence of 16-bit zero blocks may be compressed.

Depending on the platform, you can use one of two methods to obtain a list of the IPv6 addresses assigned to a computer's interfaces. For the Microsoft Research and Windows 2000 Technology Preview stacks downloaded from the Web as well as Windows XP Home Edition and Windows XP Professional, the IPV6.EXE command is used. To enumerate the IPv6 interfaces, execute IPV6.EXE if at the command prompt. For all versions of Windows 2000 and Windows XP (including future versions of Windows releases), the NETSH.EXE command may also be used. The command syntax is: NETSH.EXE interface IPv6 show interface. To programmatically obtain the configuration of local interfaces, the SIO_ADDRESS_LIST_QUERY ioctl (Chapter 7) and the IP Helper API (Chapter 16) can be used.

There are three basic types of IPv6 addresses: unicast, anycast, and multicast. Note that IPv6 does not define a broadcast address (multicasting is used instead). In the following sections, we will discuss each address type.

**Unicast**

A unicast address identifies a single interface. With IPv6, however, an interface will most likely have more than one unicast address assigned to it. There are four types of unicast addresses that you will likely encounter:

- Link-local addresses
- Site-local addresses
- Global addresses
- Compatibility addresses

An interface will always have a link-local address assigned to it—each physical network interface is auto configured with one. A link-local address is used to communicate only with other nodes on the same link. Link-local address always begin with an fe80::/64 prefix. Also, because no routing information is kept for link-local addresses, the interface index is often displayed with the address. Every physical interface on the system is assigned an adapter index number (also known as a scope ID). When a link-local address is assigned to an

interface, the link number is appended to the address. The following address is the link-local address assigned to the physical adapter whose interface index number is five.

fe80::250:8bff:fea0:92ed%5

In Winsock, if a connection is being established using link-local addresses, then the interface index must be present to indicate which link the remote host is reachable from. An IPv6 link-local address is synonymous with an IPv4 APIPA address discussed earlier in the chapter.

For example, consider host A, which has the link-local address fe80::250:8bff:fea0:92ed%5 and host B, which has the link-local address fe80::250:daff:fec3:9e34%4. If host A issues a connect to host B, it would use the destination address of B with its own scope ID that can reach host B. The address to connect to would be fe80::250:daff:fec3:9e34%5.

Site-local addresses are IPv6 addresses that are reachable only on the local network environment, such as the corporate network at a particular site. These addresses are comparable to the IPv4 private address space because they cannot be reached from other sites or the Internet and routers on the private network do not forward this traffic beyond the local site. Site-local addresses use the prefix fec0::/48. Site-local addresses must be assigned from either an IPv6 router or via DHCPv6. Currently, Microsoft's implementation of IPv6 does not support DHCPv6. IPv6-enabled routers will send Router Advertisement (RA) messages, which advertise the network portion of the address (such as the first 64 bits of the address consisting of the 48-bit site-local prefix and a 16-bit subnet ID), which the host will then use to assign a site-local address to the interface on which the RA was received.

Global addresses are just that: globally reachable on IPv6 Internet. Global addresses begin with 001. The remaining 61 bits of the first 64 bits are used to establish a routing hierarchy, and the last 64 bits comprise the interface identifier that uniquely identifies a network interface on a subnet. Global addresses are also assigned via router advertisements or by using DHCPv6.

The last type of unicast addresses are compatibility addresses, which are designed to aid in the transition from IPv4 to IPv6. There are four kinds of compatibility addresses that Windows supports: Intrasite Automatic Tunnel Addressing Protocol (ISATAP), 6to4, 6over4, and IPv4 compatible. ISATAP addresses can be derived from any IPv6 unicast address, such as link-local, site-local, and global addresses. Most often you will see an ISATAP address derived

from a link-local address. These addresses also contain an embedded IPv4 address. For example, the ISATAP address fe80::5efe:172.17.7.2 is a link-local address and contains the IPv4 address of the host (172.17.7.2). When data is sent from this interface, the IPv6 packet is encapsulated within an IPv4 header. The IPv4 destination address is obtained from the v4 address embedded within the IPv6 ISATAP destination address. The v4 address must be globally reachable for two endpoints to communicate via automatic tunneling. ISATAP addresses are currently an Internet Engineering Task Force (IETF) draft.

The second type of compatibility address is called 6to4 and is described in RFC 3056. 6to4 addresses use the global prefix 2002:WWXX:YYZZ::/48, in which WWXX:YYZZ is the hexadecimal-colon representation of w.x.y.z, a public IPv4 address. 6to4 allows IPv6/IPv4 hosts to communicate over an IPv4 routing infrastructure.

Windows XP provides a 6to4 service. This service allows hosts to communicate with other 6to4 hosts within the same site, 6to4 hosts connected to the Internet, 6to4 hosts in other sites across the IPv4 Internet, as well as with hosts on the IPv6 Internet using a 6to4 relay router. On Windows XP, the 6to4 service is configured to run automatically. If there is a public IPv4 address assigned to an interface, a 6to4 Tunneling Interface (interface index 3) is created and assigned the 6to4 address(es).

The third type of compatibility address is 6over4, which is a tunneling technique using IPv4 multicasting. It allows IPv4 and IPv6 nodes to communicate using IPv6 over an IPv4 infrastructure. This technique is described in RFC 2529.

The last type of compatibility address is the IPv4 compatible address. These addresses take the form of 0:0:0:0:0:0:w.x.y.z (or ::w.x.y.z) in which w.x.y.z is the dotted decimal representation of a public IPv4 address. When a IPv4 compatible address is used by an application as the destination, the IPv6 traffic is automatically encapsulated within an IPv4 header and sent to the destination over the IPv4 network.

**Anycast**

Anycast is an address that identifies multiple interfaces. The purpose of these addresses is to route packets destined to an anycast address to the nearest interface assigned that anycast address. A good scenario for anycast addresses is when there are several nodes on the network that provide a certain service. Each machine can be assigned the same anycast address and

clients interested in contacting that service will be routed to the nearest member. This is different from multicast because this communication is one to one of many instead of one to many. Currently however, anycast addresses are assigned to routers only.

**Multicast**

Multicasting in IPv6 is similar to IPv4 multicasting. A process joins a multicast group on a particular interface and data destined to that multicast address is received. IPv6 multicast addresses begin with 1111 1111 (FF). IPv6 multicasting and IPv6 multicast addresses are covered in more detail in Chapter 9.

### 7.5.2 IPv6 Management Protocols

IPv6 requires only a single helper protocol: Internet Control Message Protocol for IPv6 (ICMPv6), which is defined in RFC 2463. ICMPv6 provides the same types of services that ICMP does, such as destination unreachable, echo and echo reply, but also provides a mechanism for Multicast Listener Discovery (MLD) and Neighbor Discovery (ND). MLD replaces IGMP and ND replaces ARP.

### 7.5.3 Addressing IPv6 from Winsock

To specify IPv6 addresses in Winsock applications, the following structure is used.

```
struct sockaddr_in6 {
        short          sin6_family;
        u_short        sin6_port;
        u_long         sin6_flowinfo;
        struct in6_addr sin6_addr;
        u_long         sin6_scope_id;
};
```

The first field simply identifies the address family, which is AF_INET6, and the second is the port number. All fields within this structure must be in network byte order. Note that all the information discussed about port numbers in the IPv4 section apply equally to IPv6 because the port number is a property of the encapsulated protocols, such as TCP and UDP, which are also available from IPv6. The third field, sin6_flowinfo, is used to mark the traffic for the

connection but is not implemented in the Microsoft IPv6 stack. The fourth field is a 16-byte structure that contains the binary IPv6 address. The last member, sin6_scope_id, indicates the interface index (or scope ID) on which the address is located. Remember that for link-local addresses, the local scope ID on which the destination is located must be specified and the sin6_scope_id field is used for this. Site-local addresses may reference the site number as the scope ID. Global addresses do not contain a scope ID.

One last item to note is that the SOCKADDR_IN6 structure is 28 bytes in length and the SOCKADDR and SOCKADDR_IN structures are only 16 bytes long.

### 7.6 Address and Name Resolution

In this section, we'll cover how to assign both literal string addresses and resolve names to the address specific structures for both IP protocols. First, we will cover the new name resolution APIs: getaddrinfo and getnameinfo. These APIs have replaced the IPv4 specific routines. Then we'll cover the generic Winsock APIs for converting between string literal addresses and socket address structure. These APIs are WSAAddressToString and WSAStringToAddress. Note that these functions perform only address conversion and assignment, not name resolution.

Next, the IPv4 specific legacy routines will be described. We include the legacy API descriptions in case legacy code needs to be maintained, but any new projects should use the newer API functions. By using the newer functions it will be trivial to write an application that can seamlessly operate over both IPv4 and IPv6, which is the topic of the last section in this chapter.

Finally, note that all the name resolution functions covered in this chapter deal only with resolving names and not registering a name with an address. This is accomplished by the Winsock Registration and Name Resolution (RNR) APIs discussed in Chapter 8.

#### 7.6.1 Name Resolution Routines

Along with IPv6, several new name resolution functions were introduced that could handle both IPv4 and IPv6 addresses. The legacy functions like gethostbyname and inet_addr work with IPv4 addresses only. The replacement functions are named getnameinfo and getaddrinfo.

These new name resolution routines are defined in WS2TCPIP.H. Also, note that although these functions are new for Windows XP, they can be made available to work on all Winsock 2 enabled platforms. This is done by including the header file WSPIAPI.H before including WS2TCPIP.H. The compiled binary will then run on all Winsock 2–enabled platforms, such as Windows 95, Windows 98, Windows Me, Windows NT 4.0, and Windows 2000.

The getaddrinfo function provides protocol-independent name resolution. The function prototype is

```
int getaddrinfo(
                const char FAR *nodename,
                const char FAR *servname,
                const struct addrinfo FAR *hints,
                struct addrinfo FAR *FAR *res
);
```

The nodename parameter specifies the NULL-terminated host name or literal address. The servname is a NULL-terminated string containing the port number or a service name such as "ftp" or "telnet." The third parameter, hints, is a structure that can pass one or more options that affect how the name resolution is performed. Finally, the res parameter returns a linked list of addrinfo structure containing the addresses the string name was resolved to. If the operation succeeds, zero is returned; otherwise the Winsock error code is returned.

The addrinfo structure is defined as

```
struct addrinfo {
                int             ai_flags;
                int             ai_family;
                int             ai_socktype;
                int             ai_protocol;
                size_t          ai_addrlen;
                char            *ai_canonname;
                struct sockaddr *ai_addr;
                struct addrinfo *ai_next;
};
```

When passing hints into the API, the structure should be zeroed out beforehand, and the first four fields are relevant:

- ai_flags indicates one of three values: AI_PASSIVE, AI_CANONNAME, or AI_NUMERICHOST. AI_CANONNAME indicates that nodename is a computer name like www.microsoft.com and AI_NUMERICHOST indicates that it is a literal string address such as "10.10.10.1". AI_PASSIVE will be discussed later.
- ai_family can indicate AF_INET, AF_INET6, or AF_UNSPEC. If you wish to resolve to a specific address, type the supply AF_INET or AF_INET6. Otherwise, if AF_UNSPEC is given, then the addresses returned could be either IPv4 or IPv6 or both.
- ai_socktype specifies the desired socket type, such as SOCK_DGRAM, SOCK_STREAM. This field is used when servname contains the name of a service. That is, some services have different port numbers depending on whether UDP or TCP is used.
- ai_protocol specifies the desired protocol, such as IPPROTO_TCP. Again, this field is useful when servname indicates a service.

If no hints are passed into getaddrinfo, the function behaves as if a zeroed hints structure was provided with an ai_family of AF_UNSPEC.

If the function succeeds, then the resolved addresses are returned via res. If the name resolved to more than one address, then the result is a linked list chained by the ai_next field. Every address resolved from the name is indicated in ai_addr with the length of that socket address structure given in ai_addrlen. These two fields may be passed directly into bind, connect, sendto, etc.

The following code sample illustrates how to resolve a hostname along with the port number before making a TCP connection to the server.

```
SOCKET                  s;
struct addrinfo         hints,
                        *result;
int                     rc;
```

```
memset(&hints, 0, sizeof(hints));
hints.ai_flags = AI_CANONNAME;
hints.ai_family = AF_UNSPEC;
hints.ai_socktype = SOCK_STREAM;
hints.ai_protocol = IPPROTO_TCP;
rc = getaddrinfo("foobar", "5001", &hints, &result);
if (rc != 0) {
                    // unable to resolve the name
}
s = socket(result->ai_family, result->ai_socktype,
result->ai_protocol);
if (s == INVALID_SOCKET) {
                    // socket API failed
}
rc = connect(s, result->ai_addr, result->ai_addrlen);
if (rc == SOCKET_ERROR) {
                    // connect API failed
}
freeaddrinfo(result);
```

In this example, the application is resolving the hostname "foobar" and wants to establish a TCP connection to a service on port 5001. You'll also notice that this code doesn't care if the name resolved to an IPv4 or an IPv6 address. It is possible that "foobar" has both IPv4 and IPv6 addresses registered, in which case result will contain additional addrinfo structures linked by the ai_next field. If an application wanted only IPv4 addresses registered to "foobar," the hints.ai_family should be set to AF_INET. Finally, note that the information returned via res is dynamically allocated and needs to be freed by calling the freeaddrinfo API once the application is finished using the information.

Another common action that applications perform is assigning a literal string address such as "172.17.7.1" or "fe80::1234" into a socket address structure of the appropriate type. The getaddrinfo function does this by setting the AI_NUMERICHOST flag within the hints. The following code illustrates this.

```
struct addrinfo                 hints,
```

```
                                *result;
int                             rc;

memset(&hints, 0, sizeof(hints));
hints.ai_flags = AI_NUMERICHOST;
hints.ai_family = AI_UNSPEC;
hints.ai_socktype = SOCK_STREAM;
hints.ai_protocol = IPPROTO_TCP;
rc = getaddrinfo("172.17.7.1", "5001", &hints, &result);
if (rc != 0) {
                // invalid literal address
}
// Use the result
freeaddrinfo(result);
```

The literal address "172.17.7.1" will be converted to the necessary socket address structure and returned via result. Because AF_UNSPEC is passed, the API will determine the correct socket address structure (SOCKADDR_IN or SOCKADDR_IN6) required and convert the address accordingly. As before, the port field of the resulting socket address structure will be initialized to 5001.

Note that if no flags are passed as part of the hints and a literal string address is resolved, the returned structure addrinfo containing the converted address will have the AI_NUMERICHOST flags set. Likewise, if a hostname is resolved but no hints are passed, the returned structure addrinfo flag will contain AI_CANONNAME.

The last flag that can be used with getaddrinfo is AI_PASSIVE, which is used to obtain an address that can be passed to the bind function. For IPv4, this would be INADDR_ANY (0.0.0.0) and for IPv6 it would be IN6ADDR_ANY (::). To obtain the bind address, the hints should indicate which address family the passive address is to be obtained for (via ai_family), nodename should be NULL, and servname should be non-NULL—indicating the port number the application will bind to (which can be "0"). If AF_UNSPEC is passed in the hints, then two addrinfo structures will be returned, one with the IPv4 bind address and the other with the IPv6 bind address.

The AI_PASSIVE flag is useful after resolving a hostname via getaddrinfo. Once the resolved address is returned, the original result's ai_family can be used in another call to getaddrinfo to obtain the appropriate bind address for that address family. This prevents applications from having to touch the internal socket address structure's fields and also removes the need for two separate code paths for binding the socket depending on which address family the address was resolved to.

The other new name resolution API is getnameinfo, which performs the reverse of the getaddrinfo function. It takes a socket address structure already initialized and returns the host and service name corresponding to the address and port information. The getnameinfo function is prototyped as the following:

```
int getnameinfo(
                const struct sockaddr FAR *sa,
                socklen_t salen,
                char FAR *host,
                DWORD hostlen,
                char FAR *serv,
                DWORD servlen,
                int flags
);
```

The parameters are fairly self-explanatory. sa is the socket address structure on which the name information will be obtained for and salen is the size of that structure. host is the character buffer to receive the host's name. By default, the fully qualified domain name (FQDN) will be returned. hostlen simply indicates the size of the host buffer. serv is the character buffer to receive the service/port information and servlen is the length of that buffer. Finally, the flags parameter indicates how the socket address should be resolved. The possible flag values are the following:

- NI_NOFQDN indicates that only the relative distinguished name (RDN) returned. For example, with this flag set the host named "mist.microsoft.com" would return only "mist".
- NI_NUMERICHOST indicates to return the string representation of the address and not the hostname.

- NI_NAMEREQD indicates if the address cannot be resolved to a FQDN to return an error.
- NI_NUMERICSERV indicates to return the port information as a string instead of resolving to a well-known service name, such as "ftp." Note that if the serv buffer is supplied with this flag absent and the port number cannot be resolved to a well-known service, genameinfo will fail with error WSANO_DATA (11004).
- NI_DGRAM is used to differentiate datagram services from stream services. This is necessary for those few services that define different port numbers for UDP and TCP.

There is a file named RESOLVER.CPP that performs name resolution on the companion CD. A hostname or address is passed to the application along with service or port information and is resolved via getaddrinfo. Then for every resolved address returned, getnameinfo is called to obtain either the machine name back or the string literal address.

### 7.6.2 Simple Address Conversion

When an application needs only to convert between string literal addresses and socket address structures, the WSAStringToAddress and WSAAddressToString APIs are available. WSAStringToAddress is not as "smart" as getaddrinfo because you must specify the address family that the string address belongs to. The API is the following:

```
INT WSAStringToAddress(
                LPTSTR AddressString,
                INT AddressFamily,
                LPWSAPROTOCOL_INFO lpProtocolInfo,
                LPSOCKADDR lpAddress,
                LPINT lpAddressLength
);
```

The first parameter is the string to convert and the second indicates the address family the string belongs to (such as AF_INET, AF_INET6, or AF_IPX). The third parameter, lpProtocolInfo, is an optional pointer to the WSAPROTOCOL_INFO structure that defines the protocol provider to use when performing the conversion. If there are multiple providers implementing a protocol, this parameter can be used to specify an explicit provider. The

fourth parameter is the appropriate socket address structure to which the string address will be converted and assigned into.

Note that this API will convert string addresses that contain port numbers. For example, the IPv4 string notation allows a colon followed by the port number at the end of the address. For example, "157.54.126.42:1200" indicates the IPv4 address using port 1200. In IPv6, the IPv6 address string must be enclosed in square brackets after which the colon and port notation may be used. For example, [fe80::250:8bff:fea0:92ed%5]:80 indicates a link-local address with its scope ID followed by port 80. Note that only port numbers will be resolved and not service names (such as "ftp"). For both these examples, if these strings were converted with WSAStringToAddress, then the returned socket address structure will be initialized with the appropriate binary IP address, port number, and address family. For IPv6, the scope ID field will also be initialized if the string address contains "%scope_ID" after the address portion.

The WSAAddressToString provides a mapping from a socket address structure to a string representation of that address. The prototype is

```
INT WSAAddressToString(
                LPSOCKADDR lpsaAddress,
                DWORD dwAddressLength,
                LPWSAPROTOCOL_INFO lpProtocolInfo,
                LPTSTR lpszAddressString,
                LPDWORD lpdwAddressStringLength
);
```

This function takes a SOCKADDR structure and formats the binary address to a string indicated by the buffer lpszAddressString. Again, if there is more than one transport provider for a given protocol, a specific one may be selected by passing its WSAPROTOCOL_INFO structure as lpProtocolInfo. Note that the address family is a field of the SOCKADDR structure passed as lpsaAddress.

### 7.6.3 Legacy Name Resolution Routines

This section covers the legacy name resolution and is included only for the sake of code maintenance, because new applications should be using getaddrinfo and getnameinfo. The other feature you will notice is that the two new API calls replace eight legacy functions.

The function inet_addr converts a dotted IPv4 address to a 32-bit unsigned long integer quantity. The inet_addr function is defined as

```
unsigned long inet_addr(
   const char FAR *cp
);
```

The cp field is a null-terminated character string that accepts an IP address in dotted notation. Note that this function returns an IPv4 address as a 32-bit unsigned long integer in network-byte order, which can be assigned into the SOCKADDR_IN field sin_addr.

The reverse of inet_addr is inet_ntoa, which takes an IPv4 network address and converts it to a string. This function is declared as

```
char FAR *inet_ntoa(
                    Struct in_addr in
);
```

The following code sample demonstrates how to create a SOCKADDR_IN structure using the inet_addr and htons functions.

```
SOCKADDR_IN InternetAddr;
INT nPortId = 5150;

InternetAddr.sin_family = AF_INET;

// Convert the proposed dotted Internet address 136.149.3.29
// to a 4-byte integer, and assign it to sin_addr

InternetAddr.sin_addr.s_addr = inet_addr("136.149.3.29");

// The nPortId variable is stored in host-byte order. Convert
// nPortId to network-byte order, and assign it to sin_port.

InternetAddr.sin_port = htons(nPortId);
```

The Winsock functions gethostbyname, WSAAsyncGetHostByName, gethostbyaddr, and WSAAsyncGetHostByAddr retrieve host information corresponding to a host name or host address from a host database. The first two functions translate a hostname to its network IPv4 addresses and the second two do the reverse—map an IPv4 network address back to a hostname. These functions return a HOSTENT structure that is defined as

```
struct hostent
{
  char FAR *      h_name;
  char FAR * FAR * h_aliases;
  short           h_addrtype;
  short           h_length;
  char FAR * FAR * h_addr_list;
};
```

The h_name field is the official name of the host. If your network uses the DNS, it is the FQDN that causes the name server to return a reply. If your network uses a local "hosts" file, it is the first entry after the IP address. The h_aliases field is a null-terminated array of alternative names for the host. The h_addrtype represents the address family being returned. The h_length field defines the length in bytes of each address in the h_addr_list field, which will be four bytes for IPv4 addresses. The h_addr_list field is a null-terminated array of IP addresses for the host. (A host can have more than one IP address assigned to it.) Each address in the array is returned in network-byte order.

Normally, applications use the first address in the array. However, if more than one address is returned, applications should randomly choose an available address rather than always use the first address.

The prototypes for these functions are

```
struct hostent FAR * gethostbyname (
  const char FAR * name
);

HANDLE WSAAsyncGetHostByName(
```

```
    HWND hWnd,
    unsigned int wMsg,
    const char FAR * name,
    char FAR * buf,
    int buflen
);

struct HOSTENT FAR * gethostbyaddr(
    const char FAR * addr,
    int len,
    int type
);

HANDLE WSAAsyncGetHostByAddr(
                    HWND hWnd,
                    unsigned int wMsg,
                    const char FAR *addr,
                    int len,
                    int type,
                    char FAR *buf,
                    int buflen
);
```

For the first two functions, the name parameter represents a friendly name of the host you are looking for, and the latter two functions take an IPv4 network address in the addr parameter. The length of the address is specified as len. Also, type indicates the address family of the network address passed, which would be AF_INET. All four functions return the results via a HOSTENT structure. For the two synchronous functions, the HOSTENT is a system-allocated buffer that the application should not rely on being static. The two asynchronous functions will copy the HOSTENT structure to the buffer indicated by the buf parameter. This buffer size should be equal to MAXGETHOSTSTRUCT.

Finally, these and the rest of the asynchronous name and service resolution functions return a HANDLE identifying the operation issued. Upon completion, a window message indicated by

wMsg is posted to the window given by hWnd. If at some point the application wishes to cancel the asynchronous request, the WSACancelAsyncRequest function is used. This function is declared as

```
int WSACancelAsyncRequest(
                HANDLE hAsyncTaskHandle
);
```

Keep in mind that the synchronous API calls will block until the query completes or times out, which could take several seconds.

The next type of legacy resolution functions provide the capability to retrieve port numbers for well-known services and the reverse. The API functions getservbyname and WSAAsyncGetServByName take the name of a well-known service like "FTP" and return the port number that the service uses. The functions getservbyport and WSAAsyncGetServByPort perform the reverse operation by taking the port number and returning the service name that uses that port. These functions simply retrieve static information from a file named services. In Windows 95, Windows 98, and Windows Me, the services file is located under %WINDOWS%; in Windows NT, it is located under %WINDOWS%\System32\Drivers\Etc.

These four functions return the service information in a SERVENT structure that is defined as

```
struct servent {
        char FAR *          s_name;
        char FAR * FAR *s_aliases;
        short               s_port;
        char FAR *          s_proto
};
```

The field s_name is the name of the service and s_aliases is a NULL terminated array of string pointers, each containing another name for the service. s_port is the port number used by the service and s_proto is the protocol used by the service, such as the strings "tcp" and "udp."

These functions are defined as follows:

```
struct servent FAR * getservbyname(
  const char FAR * name,
  const char FAR * proto
);

HANDLE WSAAsyncGetServByName(
                HWND hWnd,
                unsigned int wMsg,
                const char FAR *name,
                const char FAR *proto,
                char FAR *buf,
                int buflen
);

struct servent FAR *getservbyport(
                int port,
                const char FAR *proto
);

HANDLE WSAAsyncGetServByPort(
                HWND hWnd,
                unsigned int wMsg,
                int port,
                const char FAR *proto,
                char FAR *buf,
                int buflen
);
```

The name parameter represents the name of the service you are looking for. The proto parameter optionally points to a string that indicates the protocol that the service in name is registered under, such as "tcp" or "udp". The second two functions simply take the port number to match to a service name. The synchronous API functions return a SERVENT structure, which is a system allocated buffer, and the asynchronous ones take an application supplied buffer, which should also be of the size MAXGETHOSTSTRUCT.

The last set of legacy name resolution API functions convert between a protocol string name, such as "tcp", and its protocol number ("tcp" would resolve to IPPROTO_TCP.). These functions are getprotobyname, WSAAsyncGetProtoByName, getprotobynumber, and WSAAsyncGetProtoByNumber. The first two convert from the string protocol to the protocol number and the latter two do the opposite—map the protocol number back to its string name. These functions return a PROTOENT structure defined as

```
struct protoent {
        char FAR *          p_name;
        char FAR * FAR *p_aliases;
        short               p_proto;
};
```

The first field, p_name, is the string name of the protocol, and p_aliases is a NULL terminated array of string pointers that contain other names the protocol is known by. Finally, p_proto is the protocol number (such as IPPROTO_UDP or IPPROTO_TCP).

These function prototypes are

```
struct protoent FAR *getprotbyname(
        const char FAR *name
);

HANDLE WSAAsyncGetProtoByName(
        HWND hWnd,
        unsigned int wMsg,
        const char FAR *name,
        char FAR *buf,
        int buflen
);

struct protoent FAR *getprotobynumber(
        int number
);
```

```
HANDLE WSAAsyncGetProtoByNumber(
                    HWND hWnd,
                    unsigned int wMsg,
                    int number,
                    char FAR *buf,
                    int buflen
);
```

These functions behave the same way the legacy name resolution functions described earlier do in terms of synchronous and asynchronous functions.

### 7.7 Writing IP Version–Independent Programs

In this section, we'll cover how to develop applications that work seamlessly over IPv4 and IPv6. This method requires using the new name resolution APIs getaddrinfo and getnameinfo and requires a bit of rearranging Winsock calls from what you are probably used to.

Before we get into the specifics, let's cover some of the basic practices that you should follow. First, applications should not allocate the socket address structures specific to each protocol (such as SOCKADDR_IN and SOCKADDR_IN6 for IPv4 and IPv6, respectively) because they can be different sizes. Instead, a new socket address structure SOCKADDR_STORAGE has been introduced that is as large as the largest possible protocol specific address structure and includes padding for 64-bit alignment issues. The following code uses a SOCKADDR_STORAGE structure to store the destination IPv6 address.

```
SOCKADDR_STORAGE                saDestination;
SOCKET                          s;
int                             addrlen,
                                rc;

s = socket(AF_INET6, SOCK_STREAM, IPPROTO_TCP);
if (s == INVALID_SOCKET) {
                    // socket failed
}
addrlen = sizeof(saDestination);
```

```
rc = WSAStringToAddress(
                        "3ffe:2900:d005:f28d:250:8bff:fea0:92ed",
                        AF_INET6,
                        NULL,
                        (SOCKADDR *)&saDestination,
                        &addrlen
                        );
if (rc == SOCKET_ERROR) {
                // conversion failed
}
rc = connect(s, (SOCKADDR *)&saDestination, sizeof(saDestination));
if (rc == SOCKET_ERROR) {
                // connect failed
}
```

Second, functions that take an address as a parameter should pass the entire socket address structure and not the protocol specific types like struct in_addr or struct in6_addr. This is important for IPv6, which might require the scope ID information to successfully connect. The SOCKADDR_STORAGE structure containing the address should be passed instead.

Third, avoid hardcode addresses regardless of whether they are IPv4 or IPv6. The Winsock header files define constants for all the address that are hard coded such as the loopback address and the wildcard address used for binding.

Now that some of the basic issues are out of the way, let's move to discussing how an application should be structured to be IP independent. We will divide our discussion into two sections: the client and the server.

**Client**

For both TCP and UDP clients, the application typically possesses the server (or recipient's) IP address or hostname. Whether it resolves to an IPv4 address or IPv6 address doesn't matter. The client should follow these three steps:

1. Resolve the address using the getaddrinfo function. The hints should contain AF_UNSPEC as well as the socket type and protocol depending on whether the client uses TCP or UDP to communicate.
2. Create the socket using the ai_family, ai_socktype, and ai_protocol fields from the addrinfo structure returned in step 1.
3. Call connect or sendto with the ai_addr member of the addrinfo structure.

The following code sample illustrates these principles.

```
SOCKET                          s;
struct addrinfo hints,
                                *res=NULL
char                            *szRemoteAddress=NULL,
                                *szRemotePort=NULL;
int                             rc;

// Parse the command line to obtain the remote server's
// hostname or address along with the port number, which are contained
// in szRemoteAddress and szRemotePort.
memset(&hints, 0, sizeof(hints));
hints.ai_family = AF_UNSPEC;
hints.ai_socktype = SOCK_STREAM;
hints.ai_protocol = IPPROTO_TCP;
// first resolve assuming string is a string literal address
rc = getaddrinfo(
                                szRemoteAddress,
                                szRemotePort,
                        &hints,
                        &res
      );
if (rc == WSANO_DATA) {
                        // Unable to resolve name - bail out
  }
s = socket(res->ai_family, res->ai_socktype, res->ai_protocol);
```

```
if (s == INVALID_SOCKET) {
                    // socket failed
}
rc = connect(s, res->ai_addr, res->ai_addrlen);
if (rc == SOCKET_ERROR) {
                    // connect failed
}
freeaddrinfo(res);
```

First, you will notice that there are no explicit references to AF_INET or AF_INET6. Also, there's no need to manipulate the underlying SOCKADDR_IN or SOCKADDR_IN6 addresses. The getaddrinfo call fully initializes the returned socket address structure with all the required information—address family, binary address, etc.—that is necessary for connecting or sending datagrams.

If the client application needs to explicitly bind the socket to a local port after socket creation but before connect or sendto, then another getaddrinfo call can be made. This call would specify the address family, socket type, and protocol returned from the first call along with the AI_PASSIVE flag and desired local port, which will return another socket address structure initialized to the necessary bind address (such as 0.0.0.0 for IPv4 and :: for IPv6).

**Server**

The server side is a bit more involved than the client side. This is because the Windows IPv6 stack is a dual stack. That is, there is a separate stack for IPv4 and IPv6, so if a server wishes to accept both IPv4 and IPv6 connections, it must create a socket for each one. The two steps for creating an IP independent server are the following:

1. Call getaddrinfo with hints containing AI_PASSIVE, AF_UNSPEC, and the desired socket type and protocol along with the desired local port to listen or receive data on. This will return two addrinfo structures: one containing the listening address for IPv4 and the other containing the listening address for IPv6.
2. For every addrinfo structure returned, create a socket with the ai_family, ai_socktype, and ai_protocol fields followed by calling bind with the ai_addr and ai_addrlen members.

The following code illustrates this principle.

```
SOCKET                          slisten[16];
char                            *szPort="5150";
struct addrinfo                 hints,
                                * res=NULL,
                                * ptr=NULL;
int                             count=0,
                                rc;

memset(&hints, 0, sizeof(hints));
hints.ai_family = AF_UNSPEC;
hints.ai_socktype = SOCK_STREAM;
hints.ai_protocol = IPPROTO_TCP;
hints.ai_flags = AI_PASSIVE;
rc = getaddrinfo(NULL, szPort, &hints, &res);
if (rc != 0) {
                // failed for some reason
}
ptr = res;
while (ptr)
{
                slisten[count] = socket(ptr->ai_family,
                ptr->ai_socktype, ptr->ai_protocol);
                if (slisten[count] == INVALID_SOCKET) {
                        // socket failed
                }
                rc = bind(slisten[count], ptr->ai_addr, ptr->ai_addrlen);
                if (rc == SOCKET_ERROR) {
                        // bind failed
                }
                rc = listen(slisten[count], 7);
                if (rc == SOCKET_ERROR) {
                        // listen failed
```

```
            }
            count++;
            ptr = ptr->ai_next;
}
```

Once the sockets are created and bound, the application simply needs to wait for incoming connections on each. Chapter 5 covers the various I/O models available in Winsock and provides fully functioning client and server samples that are written using the principles covered in this section.

## CONCLUSION

Computer networks can be used for numerous services, both for companies and for individuals. For companies, networks of personal computers using shared servers often provide access to corporate information. Typically they follow the client-server model, with client workstations on employee desktops accessing powerful servers in the machine room. For individuals, networks offer access to a variety of information and entertainment resources. Individuals often access the Internet by calling up an ISP using a modem, although increasingly many people have a fixed connection at home. An up-and-coming area is wireless networking with new applications such as mobile e-mail access and m-commerce.

Protocols are either connectionless or connection-oriented. Most networks support protocol hierarchies, with each layer providing services to the layers above it and insulating them from the details of the protocols used in the lower layers. Protocol stacks are typically based either on the OSI model or on the TCP/IP model. Both have network, transport, and application layers, but they differ on the other layers. Design issues include multiplexing, flow control, error control, and others. Much of this book deals with protocols and their design.

The present Internet is actually a collection of many thousands of networks, rather than a single network. What characterizes it is the use of the TCP/IP protocol stack throughout.

The Internet has two main transport protocols: UDP and TCP. UDP is a connectionless protocol that is mainly a wrapper for IP packets with the additional feature of multiplexing and demultiplexing multiple processes using a single IP address. UDP can be used for client-server interactions, for example, using RPC. It can also be used for building real-time protocols such as RTP.

The main Internet transport protocol is TCP. It provides a reliable bidirectional byte stream. It uses a 20-byte header on all segments. Segments can be fragmented by routers within the Internet, so hosts must be prepared to do reassembly.

## REFERENCES

### Reference to Books

[1] Andrew S. Tanenbaum, Computer Networks, Prentice Hall, March 17, 2003
[2] Andrew G. Blank, TCP/IP JumpStart-Internet Protocol Basics, 1151 Marina Village Parkway, United State of America USA,2002
[3] Anthony Jones, Jim Ohlund, Network Programming for Microsoft Windows, Redmond, Washington, 2002
[4] http://pclt.cis.yale.edu/pcll/comm/TCPIP.HTM
[5] http://www.windowsnetworking.com/articles_tutorials/lcp.html
[6] http://handshowto.com/lan101.html
[7] http://www.iso.org
[8] http://netrio.com/connecting/2000_04/