CBCS SCHEME

USN | | | | | | | | | | |

BEC/BTE702

Seventh Semester B.E./B.Tech. Degree Examination, Dec.2025/Jan.2026
Computer Networks and Protocols

Time: 3 hrs. Max. Marks: 100

Note: 1. Answer any FIVE full questions, choosing ONE full question from each module.
2. M : Marks , L: Bloom's level , C: Course outcomes.

		Module – 1	M	L	C
Q.1	a.	Explain the various types of physical topologies available in computer networks.	10	L2	CO1
	b.	With a neat diagram, explain the significance of layers in TCP/IP protocol suite.	10	L2	CO2
		OR			
Q.2	a.	Explain LAN and WAN with the help of neat diagrams.	10	L2	CO1
	b.	What is an ARP? Explain the operation of ARP and its packet format with suitable diagrams.	10	L2	CO2
		Module – 2			
Q.3	a.	Explain CSMA and show the behavior of the three persistence methods of CSMA.	10	L2	CO2
	b.	Describe flow control and error control in Data link layer.	5	L2	CO2
	c.	Write a 'C' program to perform Bit stuffing.	5	L2	CO3
		OR			
Q.4	a.	Explain CSMA/CA protocol with a flow diagram.	10	L2	CO2
	b.	Explain the Ethernet frame format of standard Ethernet.	5	L2	CO2
	c.	Write a 'C' program to perform Byte stuffing.	5	L2	CO3
		Module – 3			
Q.5	a.	Explain the working of Dynamic Host Configuration Protocol [DHCP].	10	L2	CO3
	b.	With a neat diagram, explain the virtual circuit packet network and its various phases of operation.	10	L2	CO3
		OR			
Q.6	a.	Explain IPv4 Datagram format, with a neat diagram.	10	L2	CO3
	b.	Explain distance vector routing and write a 'C' program to perform distance vector routing.	10	L2	CO3
		Module – 4			
Q.7	a.	Explain connectionless and connection oriented protocols in transport layer.	10	L2	CO2
	b.	Explain the working of Go-back-N protocol.	10	L2	CO2
		OR			
Q.8	a.	With a neat diagram, explain state transition diagram of TCP.	10	L2	CO2
	b.	Explain UDP services along with neat diagram of Pseudo header for checksum.	10	L2	CO2
		Module – 5			
Q.9	a.	Explain the following with diagram : i) WWW iii) HTTP iii) FTP.	10	L2	CO4
	b.	Explain the architecture of electronic mail with a neat diagram.	10	L2	CO4
		OR			
Q.10	a.	Explain DNS Name space, DNS in the internet and resolution.	10	L2	CO4
	b.	Explain remote logging in TELNET with a neat diagram.	10	L2	CO4

* * * * *

Q1a) Solution :

The term physical topology refers to the way in which a network is laid out physically.

Two or more devices connect to a link; two or more links form a topology.

The topology of a network is the geometric representation of the relationship of all the links and linking devices (usually called nodes) to one another.

There are four basic topologies possible: mesh, star, bus, and ring.

In a mesh topology, every device has a dedicated point-to-point link to every other device.

The term dedicated means that the link carries traffic only between the two devices it connects.

To find the number of physical links in a fully connected mesh network with n nodes, we first consider that each node must be connected to every other node.

Node 1 must be connected to n – 1 nodes, node 2 must be connected to n – 1 nodes, and finally node n must be connected to n – 1 nodes. We need n (n – 1) physical links.

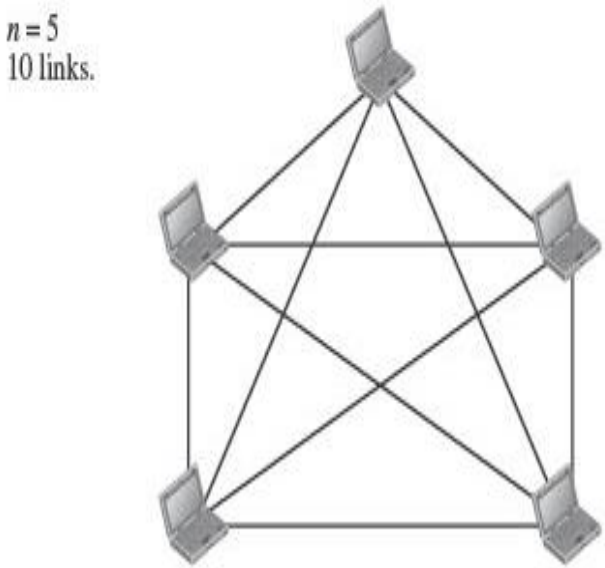
However, if each physical link allows communication in both directions (duplex mode), we can divide the number of links by 2.

In other words, we can say that in a mesh topology, we need n (n – 1) / 2 duplex-mode links.

To accommodate that many links, every device on the network must have n – 1 input/output (I/O) ports (see Figure 1.4) to be connected to the other n – 1 stations.

One practical example of a mesh topology is the connection of telephone regional offices in which each regional office needs to be connected to every other regional office.

Figure 1.4 *A fully connected mesh topology (five devices)*



In a star topology, each device has a dedicated point-to-point link only to a central controller, usually called a hub.

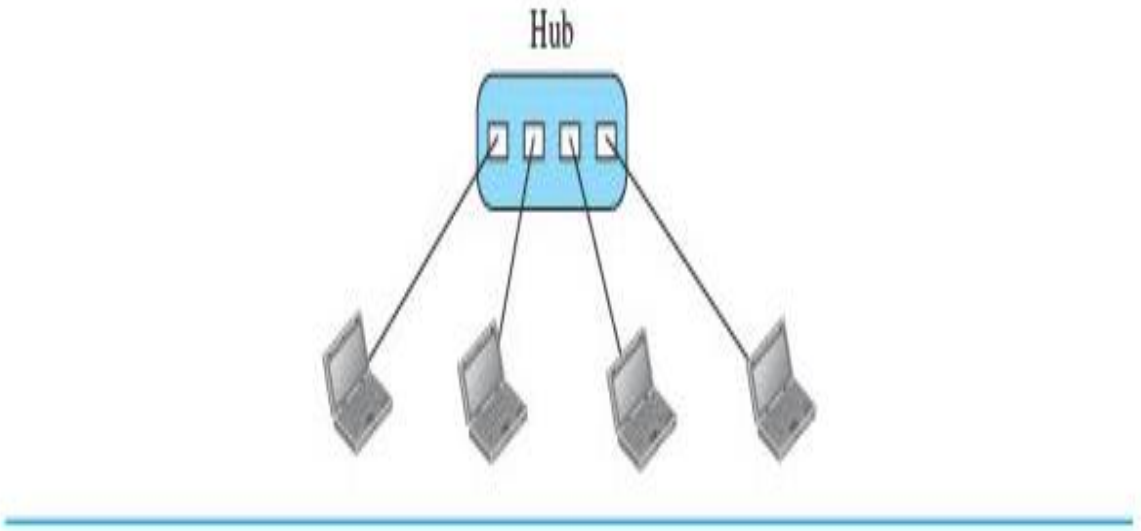
The devices are not directly linked to one another.

Unlike a mesh topology, a star topology does not allow direct traffic between devices.

The controller acts as an exchange: If one device wants to send data to another, it sends the data to the controller, which then relays the data to the other connected device (see Figure 1.5).

The star topology is used in local-area networks (LANs); High-speed LANs often use a star topology with a central hub.

Figure 1.5 A star topology connecting four stations



The preceding examples all describe point-to-point connections.

A bus topology, on the other hand, is multipoint.

One long cable acts as a backbone to link all the devices in a network (see Figure 1.6).

Nodes are connected to the bus cable by drop lines and taps.

A drop line is a connection running between the device and the main cable.

A tap is a connector that either splices into the main cable or punctures the sheathing of a cable to create a contact with the metallic core.

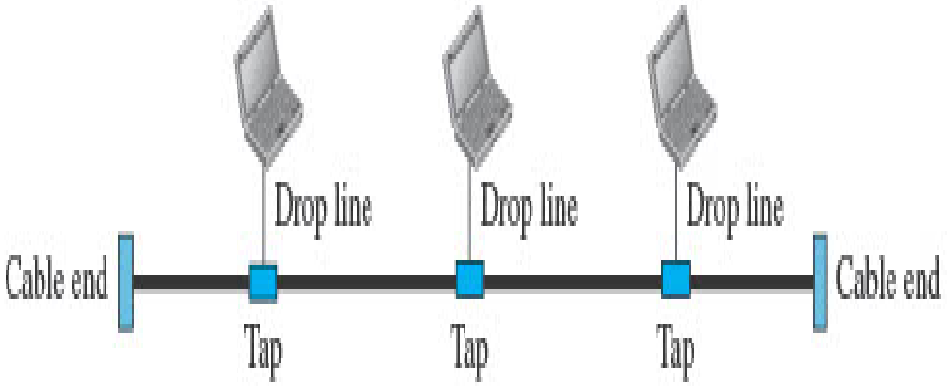
As a signal travels along the backbone, some of its energy is transformed into heat.

Therefore, it becomes weaker and weaker as it travels farther and farther.

For this reason there is a limit on the number of taps a bus can support and on the distance between those taps.

Bus topology was the one of the first topologies used in the design of early local area networks. Traditional Ethernet LANs can use a bus topology, but they are less popular now.

Figure 1.6 *A bus topology connecting three stations*



In a ring topology, each device has a dedicated point-to-point connection with only the two devices on either side of it.

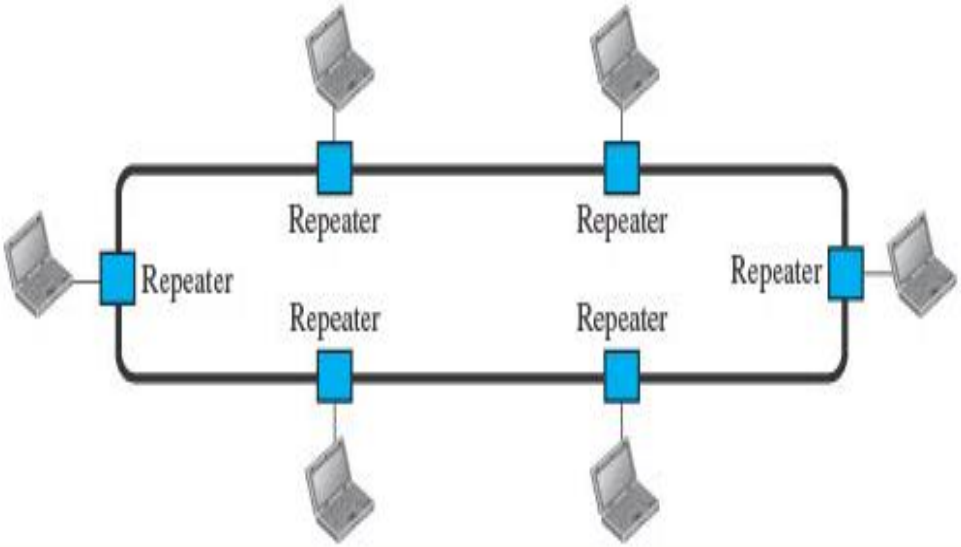
A signal is passed along the ring in one direction, from device to device, until it reaches its destination.

Each device in the ring incorporates a repeater.

When a device receives a signal intended for another device, its repeater regenerates the bits and passes them along (see Figure 1.7).

Ring topology was prevalent when IBM introduced its local-area network, Token Ring. Today, the need for higher-speed LANs has made this topology less popular.

Figure 1.7 *A ring topology connecting six stations*



Q1b) Solution :

Now that we know about the concept of protocol layering and the logical communication between layers in our second scenario, we can introduce the TCP/IP (Transmission Control Protocol/Internet Protocol).

TCP/IP is a protocol suite (a set of protocols organized in different layers) used in the Internet today.

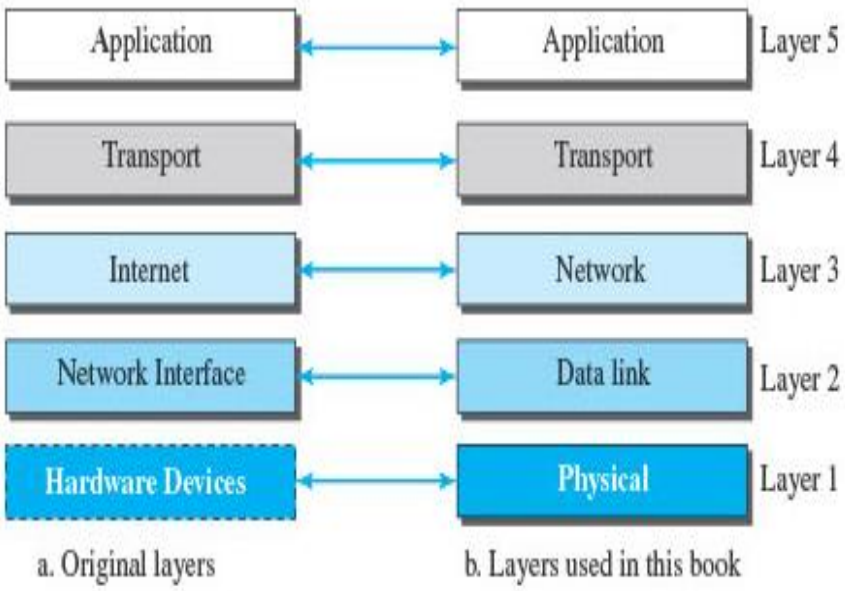
It is a hierarchical protocol made up of interactive modules, each of which provides a specific functionality.

The term hierarchical means that each upper level protocol is supported by the services provided by one or more lower level protocols.

The original TCP/IP protocol suite was defined as four software layers built upon the hardware.

Today, however, TCP/IP is thought of as a five-layer model. Figure 2.4 shows both configurations.

Figure 2.4 *Layers in the TCP/IP protocol suite*



We can say that the physical layer is responsible for carrying individual bits in a frame across the link.

Although the physical layer is the lowest level in the TCP/IP protocol suite, the communication between two devices at the physical layer is still a logical communication because there is another, hidden layer, the transmission media, under the physical layer.

Two devices are connected by a transmission medium (cable or air).

We need to know that the transmission medium does not carry bits; it carries electrical or optical signals.

So the bits received in a frame from the data-link layer are transformed and sent through the transmission media, but we can think that the logical unit between two physical layers in two devices is a bit.

There are several protocols that transform a bit to a signal.

We have seen that an internet is made up of several links (LANs and WANs) connected by routers.

There may be several overlapping sets of links that a datagram can travel from the host to the destination.

The routers are responsible for choosing the best links.

However, when the next link to travel is determined by the router, the data-link layer is responsible for taking the datagram and moving it across the link.

The link can be a wired LAN with a link-layer switch, a wireless LAN, a wired WAN, or a wireless WAN.

We can also have different protocols used with any link type.

In each case, the data-link layer is responsible for moving the packet through the link.

TCP/IP does not define any specific protocol for the data-link layer.

It supports all the standard and proprietary protocols.

Any protocol that can take the datagram and carry it through the link suffices for the network layer.

The data-link layer takes a datagram and encapsulates it in a packet called a frame.

Each link-layer protocol may provide a different service.

Some link-layer protocols provide complete error detection and correction, some provide only error correction.

The network layer is responsible for creating a connection between the source computer and the destination computer.

The communication at the network layer is host-to-host.

However, since there can be several routers from the source to the destination, the routers in the path are responsible for choosing the best route for each packet.

We can say that the network layer is responsible for host-to-host communication and routing the packet through possible routes.

Again, we may ask ourselves why we need the network layer. We could have added the routing duty to the transport layer and dropped this layer.

One reason, as we said before, is the separation of different tasks between different layers. The second reason is that the routers do not need the application and transport layers. Separating the tasks allows us to use fewer protocols on the routers.

The network layer in the Internet includes the main protocol, Internet Protocol (IP), that defines the format of the packet, called a datagram at the network layer.

IP also defines the format and the structure of addresses used in this layer.

IP is also responsible for routing a packet from its source to its destination, which is achieved by each router forwarding the datagram to the next router in its path.

IP is a connectionless protocol that provides no flow control, no error control, and no congestion control services.

This means that if any of these services is required for an application, the application should rely only on the transport-layer protocol.

The network layer also includes unicast (one-to-one) and multicast (one-to-many) routing protocols.

A routing protocol does not take part in routing (it is the responsibility of IP), but it creates forwarding tables for routers to help them in the routing process.

The network layer also has some auxiliary protocols that help IP in its delivery and routing tasks.

The Internet Control Message Protocol (ICMP) helps IP to report some problems when routing a packet.

The Internet Group Management Protocol (IGMP) is another protocol that helps IP in multitasking.

The Dynamic Host Configuration Protocol (DHCP) helps IP to get the network-layer address for a host.

The Address Resolution Protocol (ARP) is a protocol that helps IP to find the link-layer address of a host or a router when its network-layer address is given.

The logical connection at the transport layer is also end-to-end.

The transport layer at the source host gets the message from the application layer, encapsulates it in a transport layer packet (called a segment or a user datagram in different protocols) and sends it, through the logical (imaginary) connection, to the transport layer at the destination host.

In other words, the transport layer is responsible for giving services to the application layer: to get a message from an application program running on the source host and deliver it to the corresponding application program on the destination host.

We may ask why we need an end-to-end transport layer when we already have an end-to-end application layer. The reason is the separation of tasks and duties, which we discussed earlier.

The transport layer should be independent of the application layer.

In addition, we will see that we have more than one protocol in the transport layer, which means that each application program can use the protocol that best matches its requirement.

The main protocol, Transmission Control Protocol (TCP), is a connection-oriented protocol that first establishes a logical connection between transport layers at two hosts before transferring data.

It creates a logical pipe between two TCPs for transferring a stream of bytes.

TCP provides flow control (matching the sending data rate of the source host with the receiving data rate of the destination host to prevent overwhelming the destination), error control (to guarantee that the segments arrive at the destination without error and resending the corrupted ones), and congestion control to reduce the loss of segments due to congestion in the network.

The other common protocol, User Datagram Protocol (UDP), is a connectionless protocol that transmits user datagrams without first creating a logical connection. In UDP, each user datagram is an independent entity without being related to the previous or the next one (the meaning of the term connectionless).

UDP is a simple protocol that does not provide flow, error, or congestion control. Its simplicity, which means small overhead, is attractive to an application program that needs to send short messages and cannot afford the retransmission of the packets involved in TCP, when a packet is corrupted or lost.

A new protocol, Stream Control Transmission Protocol (SCTP) is designed to respond to new applications that are emerging in the multimedia.

As Figure 2.6 shows, the logical connection between the two application layers is end to-end.

The two application layers exchange messages between each other as though there were a bridge between the two layers.

However, we should know that the communication is done through all the layers.

Communication at the application layer is between two processes (two programs running at this layer).

To communicate, a process sends a request to the other process and receives a response.

Process-to-process communication is the duty of the application layer.

The application layer in the Internet includes many predefined protocols, but a user can also create a pair of processes to be run at the two hosts.

The Hypertext Transfer Protocol (HTTP) is a vehicle for accessing the World Wide Web (WWW).

The Simple Mail Transfer Protocol (SMTP) is the main protocol used in electronic mail (e-mail) service.

The File Transfer Protocol (FTP) is used for transferring files from one host to another.

The Terminal Network (TELNET) and Secure Shell (SSH) are used for accessing a site remotely.

The Simple Network Management Protocol (SNMP) is used by an administrator to manage the Internet at global and local levels.

The Domain Name System (DNS) is used by other protocols to find the network-layer address of a computer.

The Internet Group Management Protocol (IGMP) is used to collect membership in a group.

Q2)a Solution:

After defining networks in the previous section and discussing their physical structures, we need to discuss different types of networks we encounter in the world today.

The criteria of distinguishing one type of network from another is difficult and sometimes confusing.

We use a few criteria such as size, geographical coverage, and ownership to make this distinction.

After discussing two types of networks, LANs and WANs, we define switching, which is used to connect networks to form an internetwork (a network of networks).

A local area network (LAN) is usually privately owned and connects some hosts in a single office, building, or campus.

Depending on the needs of an organization, a LAN can be as simple as two PCs and a printer in someone's home office, or it can extend throughout a company and include audio and video devices.

Each host in a LAN has an identifier, an address, that uniquely defines the host in the LAN.

A packet sent by a host to another host carries both the source host's and the destination host's addresses.

In the past, all hosts in a network were connected through a common cable, which meant that a packet sent from one host to another was received by all hosts.

The intended recipient kept the packet; the others dropped the packet.

Today, most LANs use a smart connecting switch, which is able to recognize the destination address of the packet and guide the packet to its destination without sending it to all other hosts.

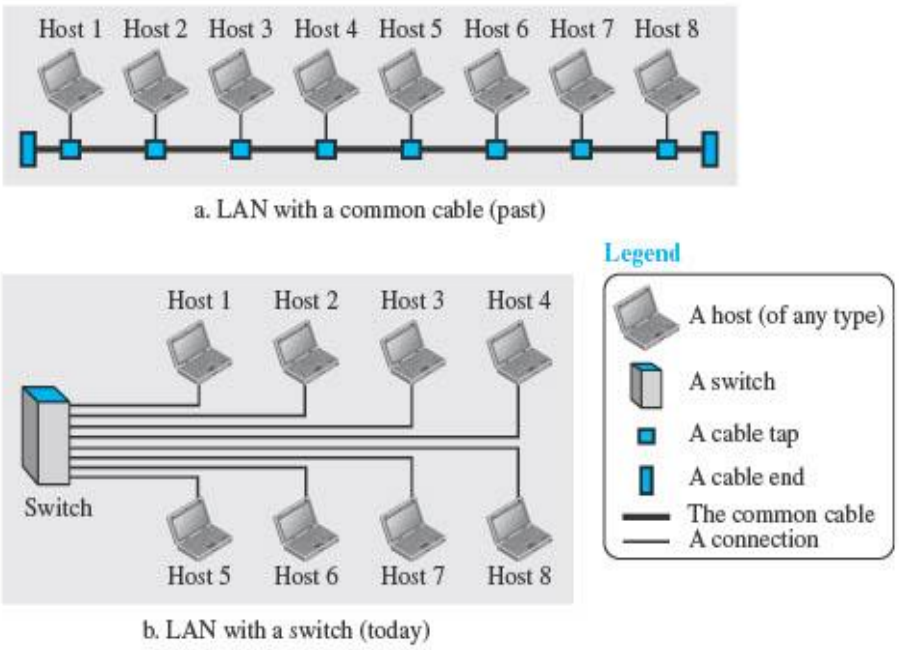
The switch alleviates the traffic in the LAN and allows more than one pair to communicate with each other at the same time if there is no common source and destination among them.

Note that the above definition of a LAN does not define the minimum or maximum number of hosts in a LAN. Figure 1.8 shows a LAN using either a common cable or a switch.

When LANs were used in isolation (which is rare today), they were designed to allow resources to be shared between the hosts.

As we will see shortly, LANs today are connected to each other and to WANs (discussed next) to create communication at a wider level.

Figure 1.8 *An isolated LAN in the past and today*



A wide area network (WAN) is also an interconnection of devices capable of communication.

However, there are some differences between a LAN and a WAN.

A LAN is normally limited in size, spanning an office, a building, or a campus; a WAN has a wider geographical span, spanning a town, a state, a country, or even the world.

A LAN interconnects hosts; a WAN interconnects connecting devices such as switches, routers, or modems.

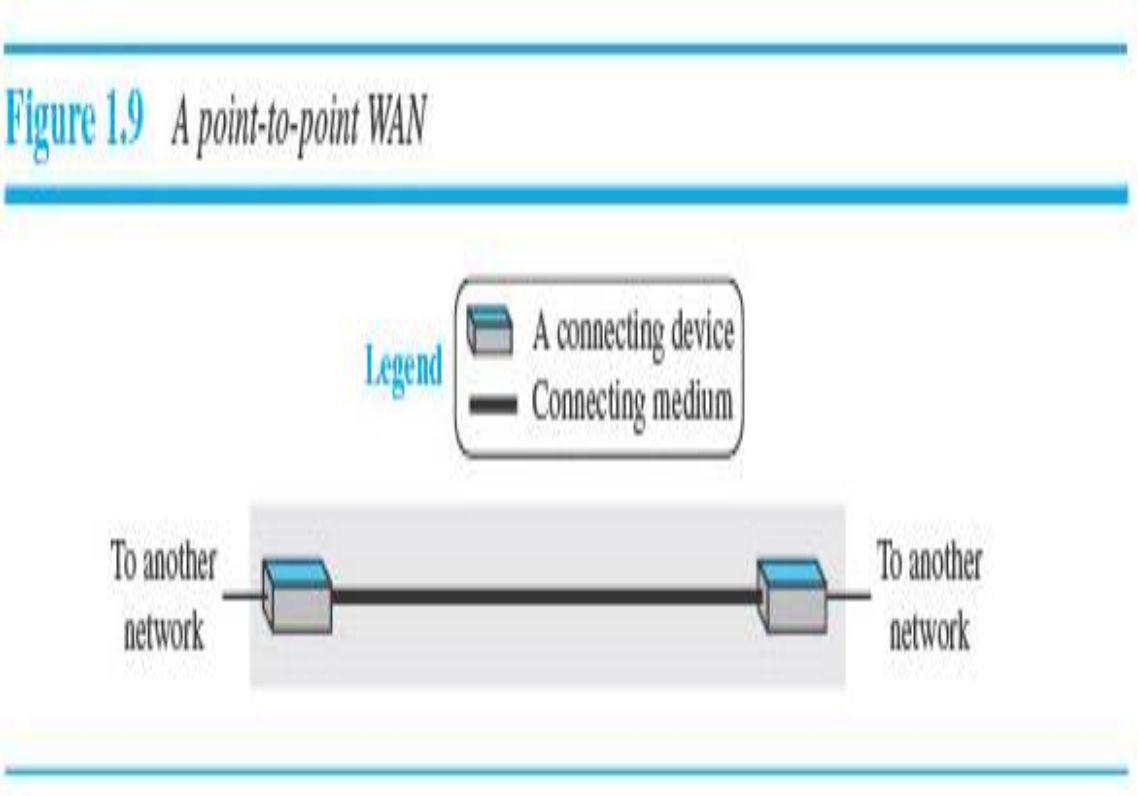
A LAN is normally privately owned by the organization that uses it; a WAN is normally created and run by communication companies and leased by an organization that uses it.

We see two distinct examples of WANs today: point-to-point WANs and switched WANs.

A point-to-point WAN is a network that connects two communicating devices through a transmission media (cable or air).

We will see examples of these WANs when we discuss how to connect the networks to one another.

Figure 1.9 shows an example of a point-to-point WAN.



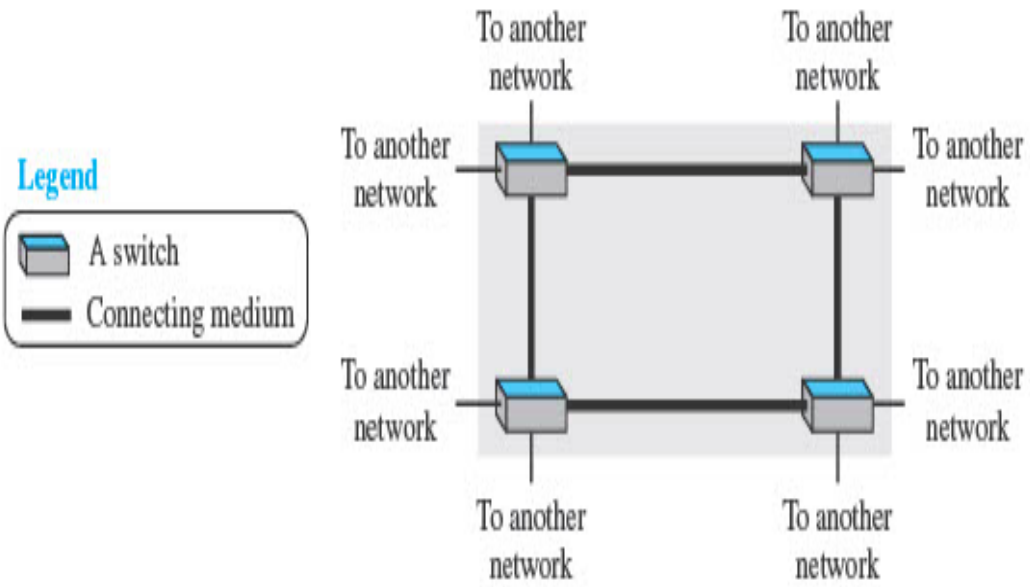
A switched WAN is a network with more than two ends.

A switched WAN, as we will see shortly, is used in the backbone of global communication today.

We can say that a switched WAN is a combination of several point-to-point WANs that are connected by switches.

Figure 1.10 shows an example of a switched WAN.

Figure 1.10 *A switched WAN*



Q2)b Solution :

Anytime a node has an IP datagram to send to another node in a link, it has the IP address of the receiving node.

The source host knows the IP address of the default router.

Each router except the last one in the path gets the IP address of the next router by using its forwarding table.

The last router knows the IP address of the destination host.

However, the IP address of the next node is not helpful in moving a frame through a link; we need the link-layer address of the next node.

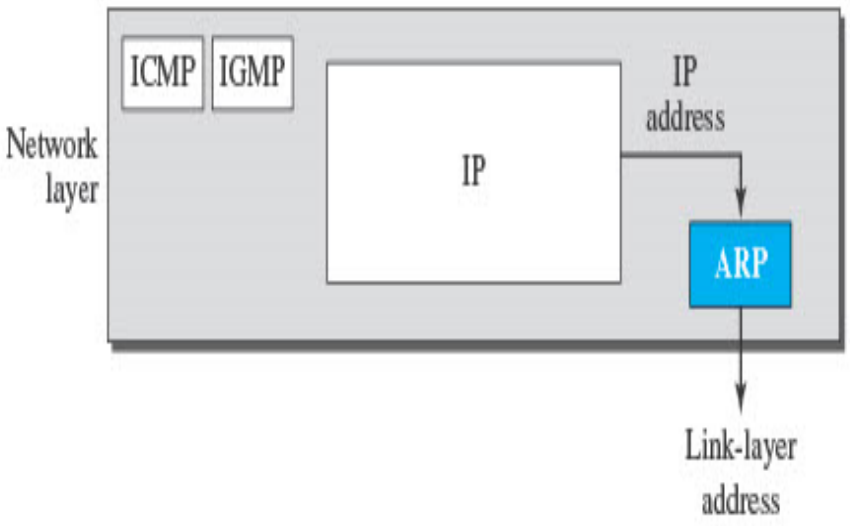
This is the time when the Address Resolution Protocol (ARP) becomes helpful.

The ARP protocol is one of the auxiliary protocols defined in the network layer, as shown in Figure 9.6.

It belongs to the network layer, but we discuss it in this chapter because it maps an IP address to a logical-link address.

ARP accepts an IP address from the IP protocol, maps the address to the corresponding link-layer address, and passes it to the data-link layer.

Figure 9.6 *Position of ARP in TCP/IP protocol suite*

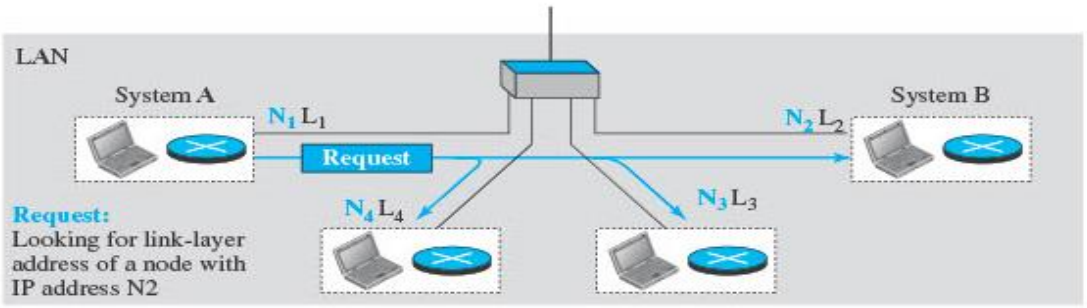


Anytime a host or a router needs to find the link-layer address of another host or router in its network, it sends an ARP request packet.

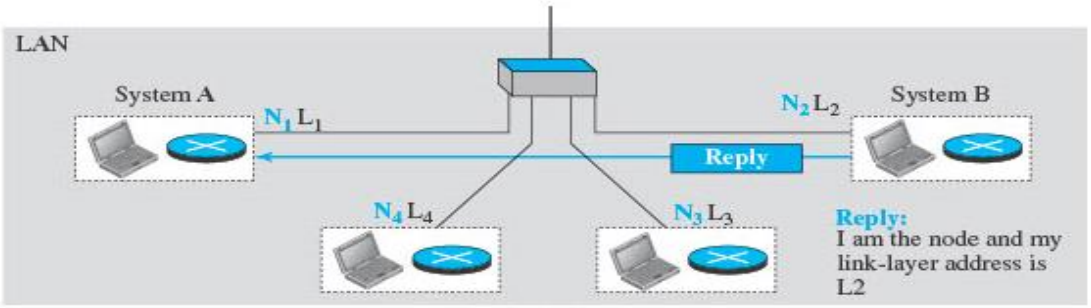
The packet includes the link-layer and IP addresses of the sender and the IP address of the receiver.

Because the sender does not know the link-layer address of the receiver, the query is broadcast over the link using the link-layer broadcast address, which we discuss for each protocol later (see Figure 9.7).

Figure 9.7 *ARP operation*



a. ARP request is broadcast



b. ARP reply is unicast

Every host or router on the network receives and processes the ARP request packet, but only the intended recipient recognizes its IP address and sends back an ARP response packet.

The response packet contains the recipient's IP and link-layer addresses.

The packet is unicast directly to the node that sent the request packet.

In Figure 9.7a, the system on the left (A) has a packet that needs to be delivered to another system (B) with IP address N2.

System A needs to pass the packet to its data-link layer for the actual delivery, but it does not know the physical address of the recipient.

It uses the services of ARP by asking the ARP protocol to send a broadcast ARP request packet to ask for the physical address of a system with an IP address of N2.

This packet is received by every system on the physical network, but only system B will answer it, as shown in Figure 9.7b.

System B sends an ARP reply packet that includes its physical address.

Now system A can send all the packets it has for this destination using the physical address it received.

Figure 9.8 shows the format of an ARP packet.

The names of the fields are self explanatory.

The hardware type field defines the type of the link-layer protocol; Ethernet is given the type 1.

The protocol type field defines the network-layer protocol: IPv4 protocol is (0800)16.

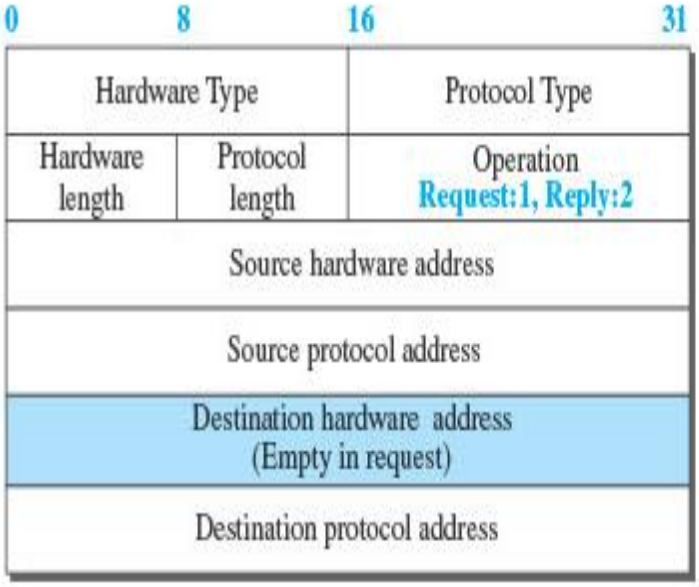
The source hardware and source protocol addresses are variable-length fields defining the link-layer and network-layer addresses of the sender.

The destination hardware address and destination protocol address fields define the receiver link-layer and network-layer addresses.

An ARP packet is encapsulated directly into a data-link frame.

The frame needs to have a field to show that the payload belongs to the ARP and not to the network-layer datagram.

Figure 9.8 *ARP packet*



Hardware: LAN or WAN protocol
Protocol: Network-layer protocol

Q3)a Solution :

To minimize the chance of collision and, therefore, increase the performance, the CSMA method was developed.

The chance of collision can be reduced if a station senses the medium before trying to use it.

Carrier sense multiple access (CSMA) requires that each station first listen to the medium (or check the state of the medium) before sending.

In other words, CSMA is based on the principle “sense before transmit” or “listen before talk.”

CSMA can reduce the possibility of collision, but it cannot eliminate it.

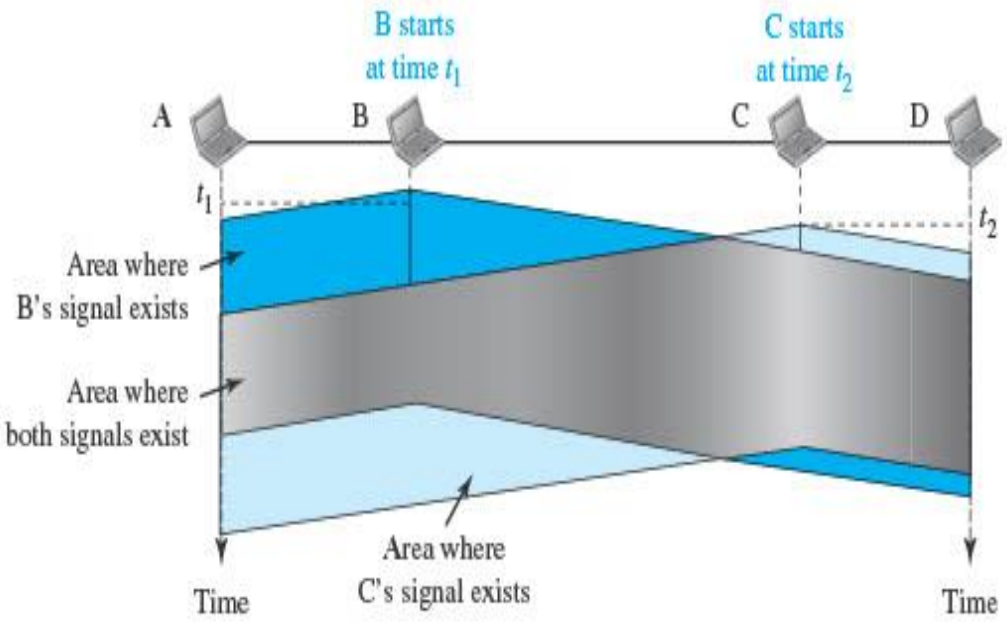
The reason for this is shown in Figure 12.7, a space and time model of a CSMA network.

Stations are connected to a shared channel (usually a dedicated medium).

The possibility of collision still exists because of propagation delay; when a station sends a frame, it still takes time (although very short) for the first bit to reach every station and for every station to sense it.

In other words, a station may sense the medium and find it idle, only because the first bit sent by another station has not yet been received.

Figure 12.7 *Space/time model of a collision in CSMA*



At time t1, station B senses the medium and finds it idle, so it sends a frame.

At time t2 (t2 > t1), station C senses the medium and finds it idle because, at this time, the first bits from station B have not reached station C.

Station C also sends a frame. The two signals collide and both frames are destroyed.

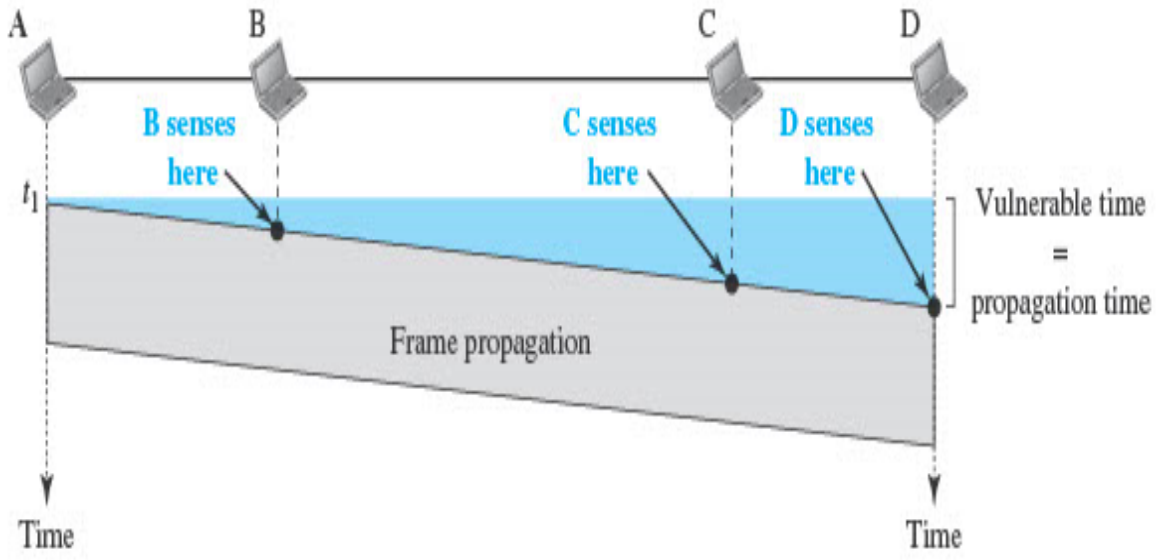
The vulnerable time for CSMA is the propagation time Tp. This is the time needed for a signal to propagate from one end of the medium to the other.

When a station sends a frame and any other station tries to send a frame during this time, a collision will result.

But if the first bit of the frame reaches the end of the medium, every station will already have heard the bit and will refrain from sending.

Figure 12.8 shows the worst case. The leftmost station, A, sends a frame at time t1, which reaches the rightmost station, D, at time t1 + Tp. The gray area shows the vulnerable area in time and space.

Figure 12.8 *Vulnerable time in CSMA*



Persistence Methods

What should a station do if the channel is busy?

What should a station do if the channel is idle?

Three methods have been devised to answer these questions: the 1-persistent method, the nonpersistent method, and the p-persistent method.

Figure 12.9 shows the behavior of three persistence methods when a station finds a channel busy.

Figure 12.9 *Behavior of three persistence methods*

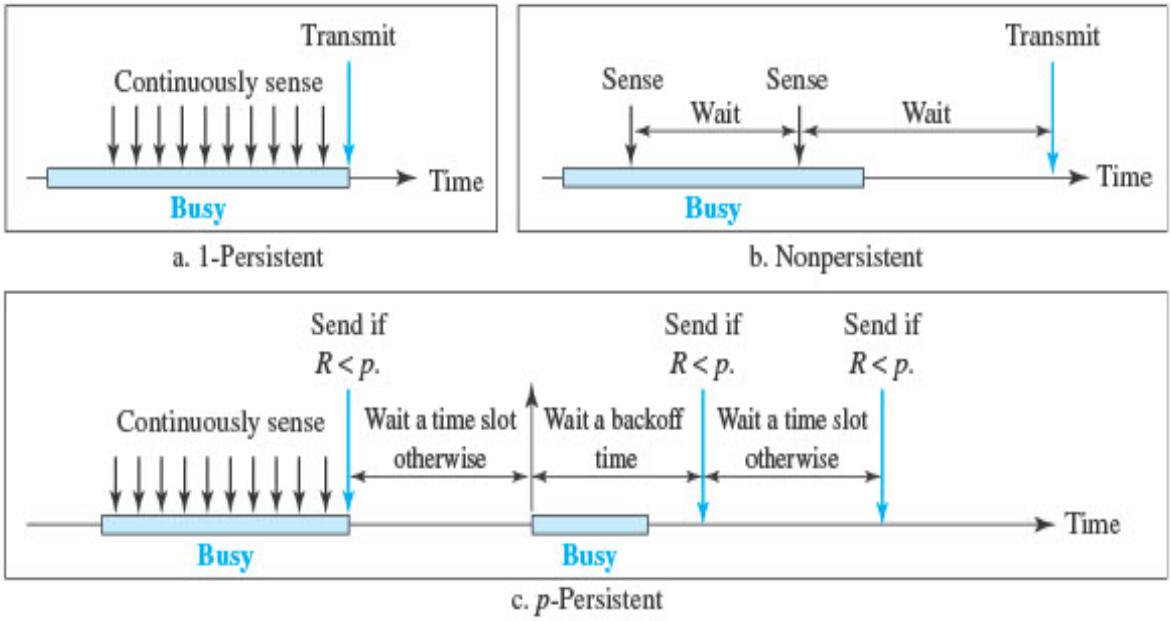


Figure 12.10 shows the flow diagrams for these methods.

1-Persistent: The 1-persistent method is simple and straightforward.

In this method, after the station finds the line idle, it sends its frame immediately (with probability 1).

This method has the highest chance of collision because two or more stations may find the line idle and send their frames immediately.

We will see later that Ethernet uses this method.

Nonpersistent: In the nonpersistent method, a station that has a frame to send senses the line.

If the line is idle, it sends immediately. If the line is not idle, it waits a random amount of time and then senses the line again.

The nonpersistent approach reduces the chance of collision because it is unlikely that two or more stations will wait the same amount of time and retry to send simultaneously.

However, this method reduces the efficiency of the network because the medium remains idle when there may be stations with frames to send.

p-Persistent: The p-persistent method is used if the channel has time slots with a slot duration equal to or greater than the maximum propagation time.

The p-persistent approach combines the advantages of the other two strategies.

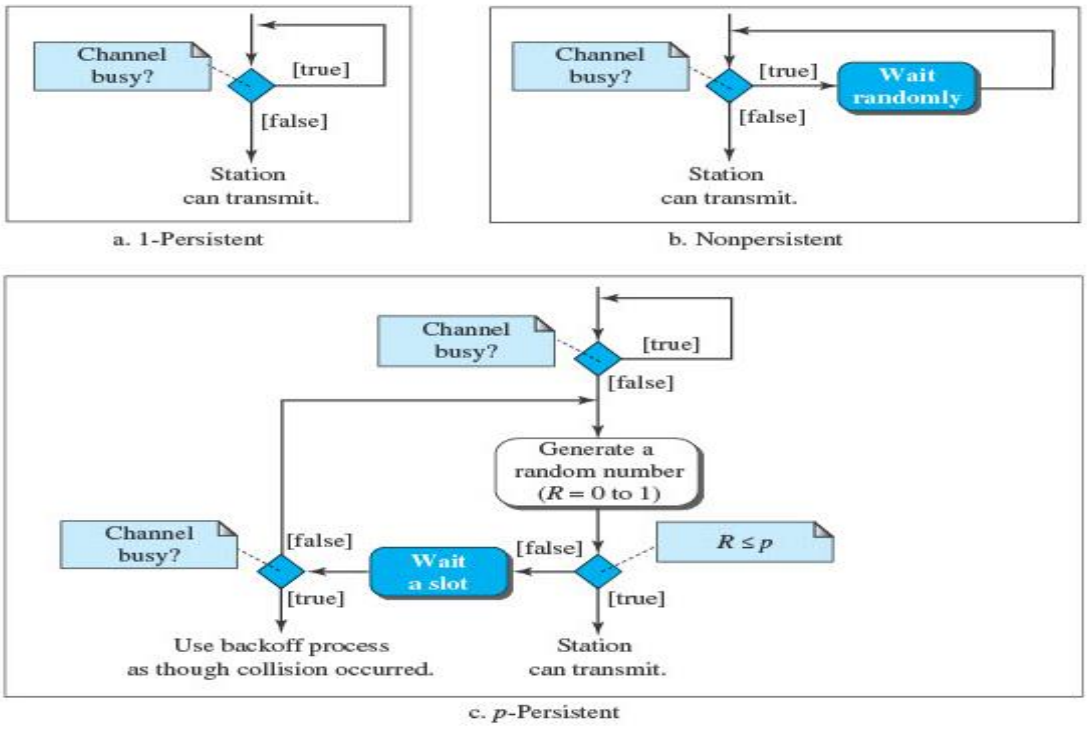
It reduces the chance of collision and improves efficiency.

In this method, after the station finds the line idle it follows these steps:

With probability p, the station sends its frame.

With probability q = 1 − p, the station waits for the beginning of the next time slot and checks the line again. a. If the line is idle, it goes to step 1. b. If the line is busy, it acts as though a collision has occurred and uses the backoff procedure.

Figure 12.10 *Flow diagram for three persistence methods*



Q3)b Solution :

We briefly defined flow and error control in Chapter 9; we elaborate on these two issues here.

One of the responsibilities of the data-link control sublayer is flow and error control at the data-link layer.

Flow Control

Whenever an entity produces items and another entity consumes them, there should be a balance between production and consumption rates.

If the items are produced faster than they can be consumed, the consumer can be overwhelmed and may need to discard some items.

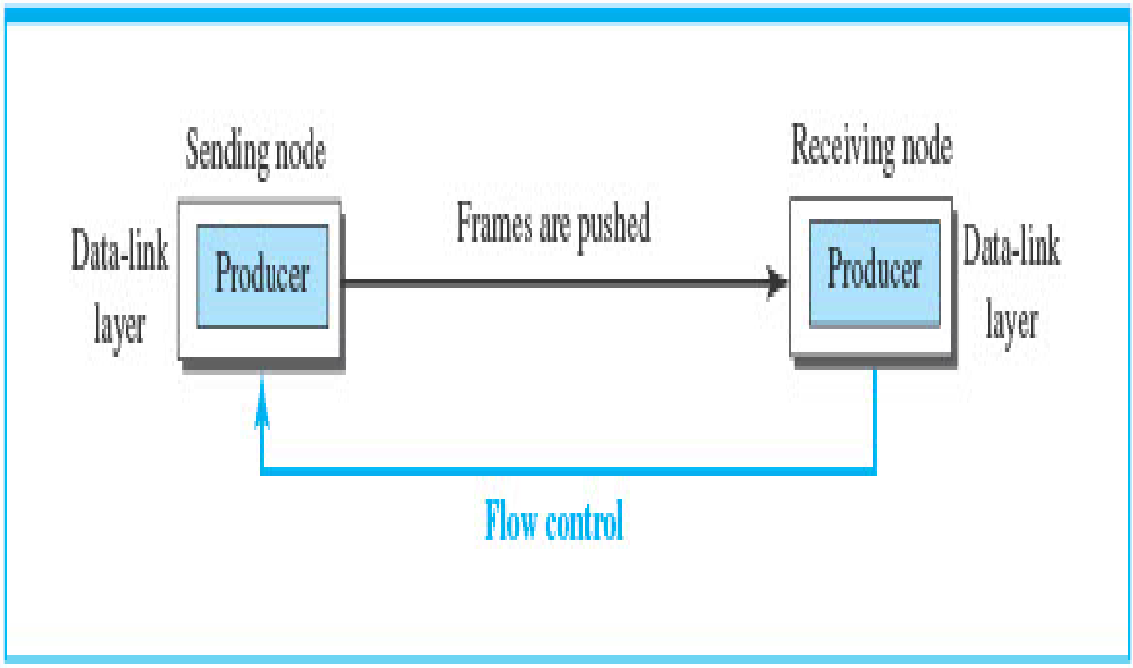
If the items are produced more slowly than they can be consumed, the consumer must wait, and the system becomes less efficient.

Flow control is related to the first issue. We need to prevent losing the data items at the consumer site.

In communication at the data-link layer, we are dealing with four entities: network and data-link layers at the sending node and network and data-link layers at the receiving node.

Although we can have a complex relationship with more than one producer and consumer , we ignore the relationships between networks and data-link layers and concentrate on the relationship between two data-link layers, as shown in Figure 11.5.

Figure 11.5 *Flow control at the data-link layer*



The figure shows that the data-link layer at the sending node tries to push frames toward the data-link layer at the receiving node.

If the receiving node cannot process and deliver the packet to its network at the same rate that the frames arrive, it becomes overwhelmed with frames.

Flow control in this case can be feedback from the receiving node to the sending node to stop or slow down pushing frames.

Error Control

Since the underlying technology at the physical layer is not fully reliable, we need to implement error control at the data-link layer to prevent the receiving node from delivering corrupted packets to its network layer.

Error control at the data-link layer is normally very simple and implemented using one of the following two methods.

In both methods, a CRC is added to the frame header by the sender and checked by the receiver.

In the first method, if the frame is corrupted, it is silently discarded; if it is not corrupted, the packet is delivered to the network layer. This method is used mostly in wired LANs such as Ethernet.

In the second method, if the frame is corrupted, it is silently discarded; if it is not corrupted, an acknowledgment is sent (for the purpose of both flow and error control) to the sender.

Flow and error control can be combined.

In a simple situation, the acknowledgment that is sent for flow control can also be used for error control to tell the sender the packet has arrived uncorrupted.

The lack of acknowledgment means that there is a problem in the sent frame.

We show this situation when we discuss some simple protocols in the next section.

A frame that carries an acknowledgment is normally called an ACK to distinguish it from the data frame.

Q3)c Solution :

```
#include <stdio.h>
```

```
#include <string.h>
#define MAX 1000

int main()
{
   int si = 0, di = 0, count = 0; /* si = source index, di = destination index*/
   char str[MAX], dest[MAX];
   char flag[MAX] = "01111110";

   printf("Enter the user data in 1's & 0's :");
   scanf("%s", str);

   di=strlen(flag); //get the string length of 'flag'
   strcpy(dest,flag); //copy 'flag' to 'dest'

   //start of stuffing
   while(str[si] != '\0')
         {
            if(str[si] == '1')
                     count++;
            else
                     count = 0; // restart the counter

            dest[di++] = str[si++];

            if(count == 5)
                  {
                     count = 0; // restart the counter
                     dest[di++] = '0'; //bit stuffed
```

```
            }
        } // end of while loop

        dest[di] = '\0'; //add null char to the end of 'dest'
        strcat(dest, flag); //concatenate 'dest' & 'flag'

        printf("Stuffed message is :%s", dest);

        // end of stuffing

        di = strlen(dest) - strlen(flag);
        dest[di] = '\0'; // to remove trailing flag and to show the end of string
// to remove flags
        for(di = strlen(flag), si = 0; dest[di]!='\0'; si++, di++)
                str[si] = dest[di];

        str[si] = '\0'; //add null char to the end of 'str'

        //start of destuffing
        si = 0, di = 0, count = 0;

        while(str[si] != '\0')
        {
                if(str[si] == '1')
                        count++;
                else
                        count = 0; // restart the counter

                dest[di++] = str[si++];
```

```
            if(count == 5)
            {
                    count = 0; // restart the counter
                    si++; //skip the bit '0' that was stuffed
            }
        } // end of while loop

        dest[di] = '\0';
        //end of destuffing

        printf("\nDestuffed message is          :%s", dest);


}
```

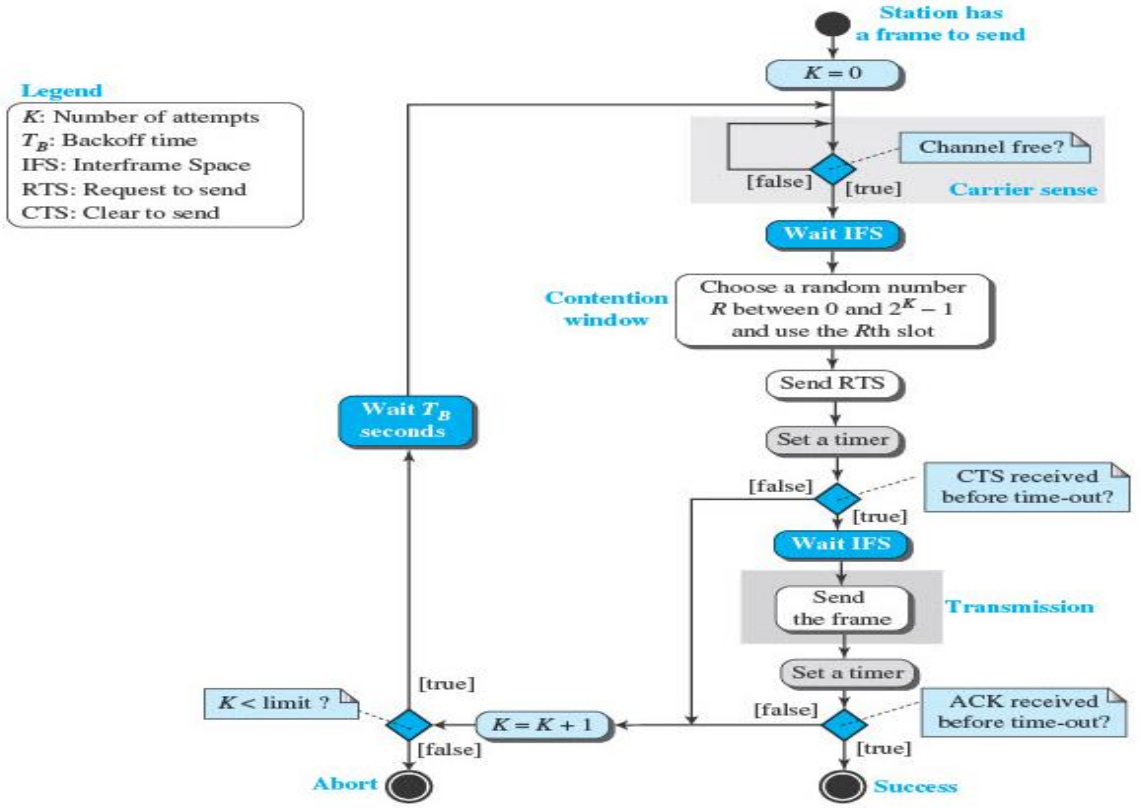
Q4)a Solution :

Carrier sense multiple access with collision avoidance (CSMA/CA) was invented for wireless networks.

Collisions are avoided through the use of CSMA/CA's three strategies: the interframe space, the contention window, and acknowledgments, as shown in Figure 12.15.

We discuss RTS and CTS frames later.

Figure 12.15 *Flow diagram of CSMA/CA*



Interframe Space (IFS): First, collisions are avoided by deferring transmission even if the channel is found idle.

When an idle channel is found, the station does not send immediately.

It waits for a period of time called the interframe space or IFS.

Even though the channel may appear idle when it is sensed, a distant station may have already started transmitting.

The distant station's signal has not yet reached this station.

The IFS time allows the front of the transmitted signal by the distant station to reach this station.

After waiting an IFS time, if the channel is still idle, the station can send, but it still needs to wait a time equal to the contention window (described next).

The IFS variable can also be used to prioritize stations or frame types.

For example, a station that is assigned a shorter IFS has a higher priority.

Contention Window: The contention window is an amount of time divided into slots.

A station that is ready to send chooses a random number of slots as its wait time. The number of slots in the window changes according to the binary exponential backoff strategy.

This means that it is set to one slot the first time and then doubles each time the station cannot detect an idle channel after the IFS time.

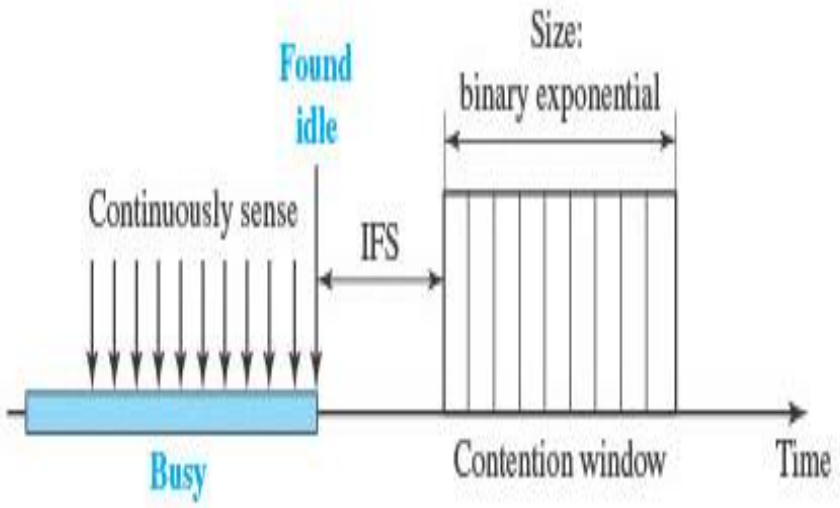
This is very similar to the p-persistent method except that a random outcome defines the number of slots taken by the waiting station.

One interesting point about the contention window is that the station needs to sense the channel after each time slot.

However, if the station finds the channel busy, it does not restart the process; it just stops the timer and restarts it when the channel is sensed as idle.

This gives priority to the station with the longest waiting time. See Figure 12.16.

Figure 12.16 *Contention window*



Acknowledgment: With all these precautions, there still may be a collision resulting in destroyed data.

In addition, the data may be corrupted during the transmission.

The positive acknowledgment and the time-out timer can help guarantee that the receiver has received the frame.

Q4)b Solution :

We refer to the original Ethernet technology with the data rate of 10 Mbps as the Standard Ethernet.

Although most implementations have moved to other technologies in the Ethernet evolution, there are some features of the Standard Ethernet that have not changed during the evolution.

We discuss this standard version to pave the way for understanding the other three technologies.

Let us first discuss some characteristics of the Standard Ethernet.

The Ethernet frame contains seven fields, as shown in Figure 13.3.

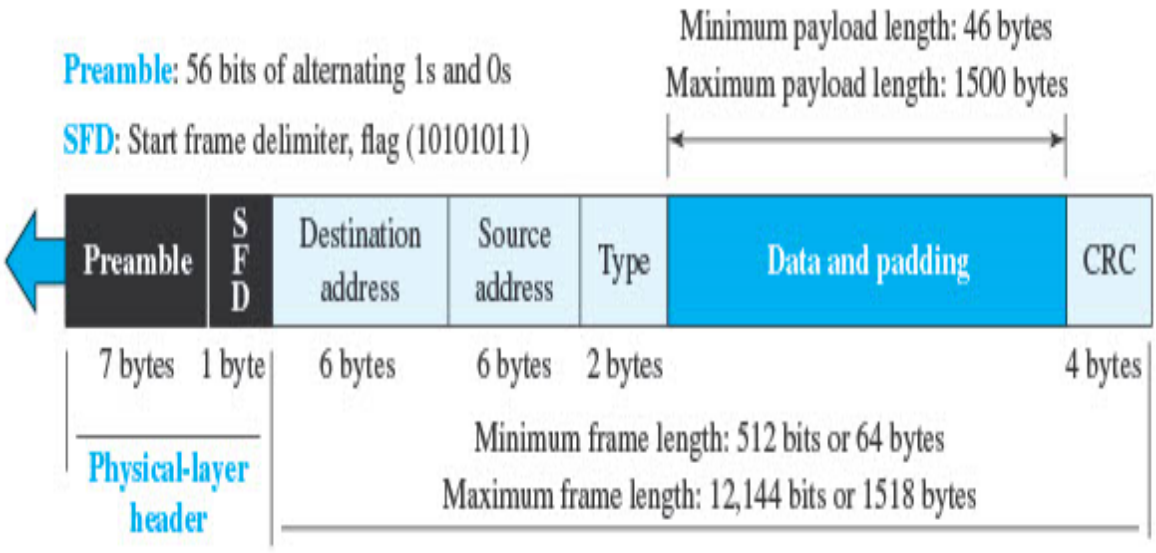
Preamble: This field contains 7 bytes (56 bits) of alternating 0s and 1s that alert the receiving system to the coming frame and enable it to synchronize its clock if it’s out of synchronization.

The pattern provides only an alert and a timing pulse.

The 56-bit pattern allows the stations to miss some bits at the beginning of the frame.

The preamble is actually added at the physical layer and is not (formally) part of the frame.

Figure 13.3 *Ethernet frame*



Start frame delimiter (SFD): This field (1 byte: 10101011) signals the beginning of the frame.

The SFD warns the station or stations that this is the last chance for synchronization.

The last 2 bits are (11)2 and alert the receiver that the next field is the destination address.

This field is actually a flag that defines the beginning of the frame.

We need to remember that an Ethernet frame is a variable-length frame.

It needs a flag to define the beginning of the frame.

The SFD field is also added at the physical layer.

Destination address (DA): This field is six bytes (48 bits) and contains the link-layer address of the destination station or stations to receive the packet.

We will discuss addressing shortly.

When the receiver sees its own link-layer address, or a multicast address for a group that the receiver is a member of, or a broadcast address, it decapsulates the data from the frame and passes the data to the upper layer protocol defined by the value of the type field.

Source address (SA): This field is also six bytes and contains the link-layer address of the sender of the packet. We will discuss addressing shortly.

Type: This field defines the upper-layer protocol whose packet is encapsulated in the frame.

This protocol can be IP, ARP, OSPF, and so on.

In other words, it serves the same purpose as the protocol field in a datagram and the port number in a segment or user datagram.

It is used for multiplexing and demultiplexing.

Data: This field carries data encapsulated from the upper-layer protocols.

It is a minimum of 46 and a maximum of 1500 bytes. We discuss the reason for these minimum and maximum values shortly.

If the data coming from the upper layer is more than 1500 bytes, it should be fragmented and encapsulated in more than one frame.

If it is less than 46 bytes, it needs to be padded with extra 0s.

A padded data frame is delivered to the upper-layer protocol as it is (without removing the padding), which means that it is the responsibility of the upper layer to remove or, in the case of the sender, to add the padding.

The upper-layer protocol needs to know the length of its data. For example, a datagram has a field that defines the length of the data.

CRC: The last field contains error detection information, in this case a CRC-32.

The CRC is calculated over the addresses, types, and data field.

If the receiver calculates the CRC and finds that it is not zero (corruption in transmission), it discards the frame.

Q4)c Solution :

```
#include <stdio.h>
#include <string.h>
#define MAX 1000

int main()
{
   int si = 0, di = 0; /* si = source index, di = destination index*/
   char src[MAX], dest[MAX];
   char flag1[] = "DLESTX", flag2[] = "DLEETX";

   //clrscr(); //clear the screen
   printf("Enter the STRING data   : ");
   scanf("%s", src);

   di=strlen(flag1); //get the string length of 'flag1'
   strcpy(dest,flag1); //copy 'flag1' to 'dest'

   //start of stuffing
```

```
  while(src[si] != '\0')
        {
          if(src[si] == 'D' && src[si+1] == 'L' && src[si+2] == 'E')
          {
                        dest[di] = 'D', dest[di+1] = 'L', dest[di+2] = 'E', dest[di+3] = 'D',
dest[di+4] = 'L', dest[di+5] = 'E';
                        di +=6;// update dest index
                        si+=3; // update source index
          }
          else
                        dest[di++] = src[si++];
        } // end of while loop

        dest[di] = '\0'; //add null char to the end of 'dest'
        strcat(dest, flag2); //concatenate 'dest' & 'flag2'

        printf("Stuffed message is      : %s", dest);

        // end of stuffing

        di = strlen(dest) - strlen(flag2);
        dest[di] = '\0'; // to remove trailing flag

        for(di = strlen(flag1), si = 0; dest[di]!='\0'; si++, di++)
                src[si] = dest[di];

        src[si] = '\0'; //add null char to the end of 'src'

        //start of destuffing
```

```
        si = 0, di = 0;

        while(src[si] != '\0')
        {
                if(src[si] == 'D' && src[si+1] == 'L' && src[si+2] == 'E')
          {
                        dest[di] = 'D', dest[di+1] = 'L', dest[di+2] = 'E';
                        si +=6; // update source index by skipping the three positions of
DLE
                        di+=3; // update dest index
          }
          else
                        dest[di++] = src[si++];
        } // end of while loop

        dest[di] = '\0';
        //end of destuffing

        printf("\nDestuffed message is    : %s", dest);


}
```

Q5)a Solution :

We have seen that a large organization or an ISP can receive a block of addresses directly from ICANN and a small organization can receive a block of addresses from an ISP.

After a block of addresses are assigned to an organization, the network administration can manually assign addresses to the individual hosts or routers.

However, address assignment in an organization can be done automatically using the Dynamic Host Configuration Protocol (DHCP).

DHCP is an application-layer program, using the client-server paradigm, that actually helps TCP/IP at the network layer.

DHCP has found such widespread use in the Internet that it is often called a plug and-play protocol.

It can be used in many situations.

A network manager can configure DHCP to assign permanent IP addresses to the host and routers.

DHCP can also be configured to provide temporary, on demand, IP addresses to hosts.

The second capability can provide a temporary IP address to a traveller to connect her laptop to the Internet while she is staying in the hotel.

It also allows an ISP with 1000 granted addresses to provide services to 4000 households, assuming not more than one-forth of customers use the Internet at the same time.

In addition to its IP address, a computer also needs to know the network prefix (or address mask).

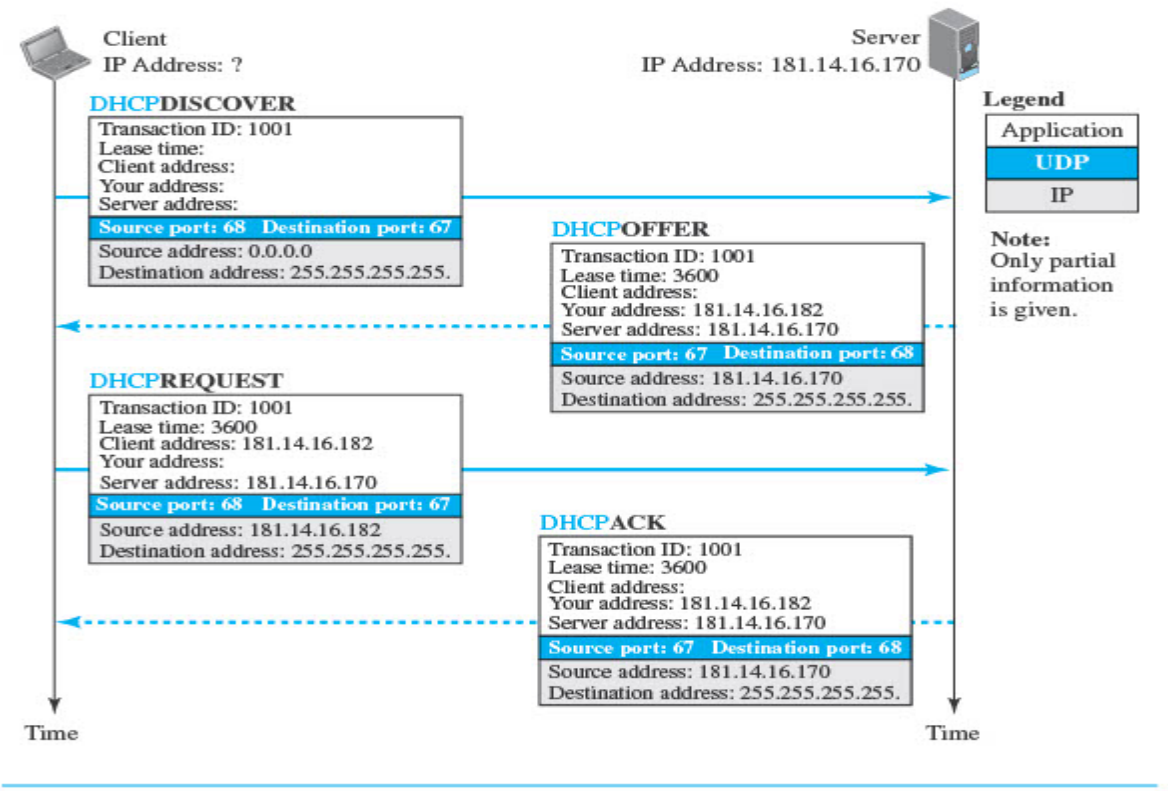
Most computers also need two other pieces of information, such as the address of a default router to be able to communicate with other networks and the address of a name server to be able to use names instead of addresses, as we will see in Chapter 26.

In other words, four pieces of information are normally needed: the computer address, the prefix, the address of a router, and the IP address of a name server.

DHCP can be used to provide these pieces of information to the host.

Figure 18.27 shows a simple scenario.

Figure 18.27 *Operation of DHCP*



1. The joining host creates a DHCPDISCOVER message in which only the transaction-ID field is set to a random number.

No other field can be set because the host has no knowledge with which to do so.

This message is encapsulated in a UDP user datagram with the source port set to 68 and the destination port set to 67.

We will discuss the reason for using two well-known port numbers later.

The user datagram is encapsulated in an IP datagram with the source address set to 0.0.0.0 (“this host”) and the destination address set to 255.255.255.255 (broadcast address).

The reason is that the joining host knows neither its own address nor the server address.

2. The DHCP server or servers (if more than one) responds with a DHCPOFFER message in which the your address field defines the offered IP address for the joining host and the server address field includes the IP address of the server.

The message also includes the lease time for which the host can keep the IP address.

This message is encapsulated in a user datagram with the same port numbers, but in the reverse order.

The user datagram in turn is encapsulated in a datagram with the server address as the source IP address, but the destination address is a broadcast address, in which the server allows other DHCP servers to receive the offer and give a better offer if they can.

3. The joining host receives one or more offers and selects the best of them.

The joining host then sends a DHCPREQUEST message to the server that has given the best offer.

The fields with known value are set.

The message is encapsulated in a user datagram with port numbers as the first message.

The user datagram is encapsulated in an IP datagram with the source address set to the new client address, but the destination address still is set to the broadcast address to let the other servers know that their offer was not accepted.

4. Finally, the selected server responds with a DHCPACK message to the client if the offered IP address is valid.

If the server cannot keep its offer (for example, if the address is offered to another host in between), the server sends a DHCPNACK message and the client needs to repeat the process.

This message is also broadcast to let other servers know that the request is accepted or rejected.

Q5)b Solution :

In a connection-oriented service (also called virtual-circuit approach), there is a relationship between all packets belonging to a message.

Before all datagrams in a message can be sent, a virtual connection should be set up to define the path for the datagrams.

After connection setup, the datagrams can all follow the same path.

In this type of service, not only must the packet contain the source and destination addresses, it must also contain a flow label, a virtual circuit identifier that defines the virtual path the packet should follow.

Shortly, we will show how this flow label is determined, but for the moment, we assume that the packet carries this label.

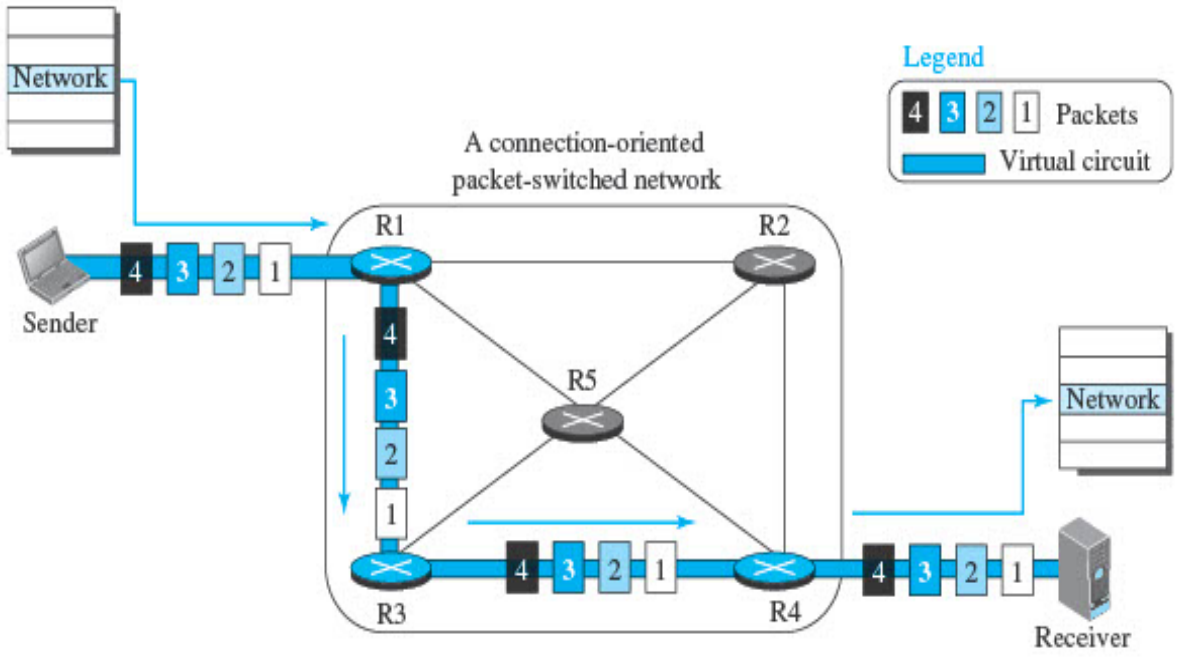
Although it looks as though the use of the label may make the source and destination addresses unnecessary during the data transfer phase, parts of the Internet at the network layer still keep these addresses.

One reason is that part of the packet path may still be using the connectionless service.

Another reason is that the protocol at the network layer is designed with these addresses, and it may take a while before they can be changed.

Figure 18.5 shows the concept of connection-oriented service.

Figure 18.5 *A virtual-circuit packet-switched network*



Each packet is forwarded based on the label in the packet.

To follow the idea of connection-oriented design to be used in the Internet, we assume that the packet has a label when it reaches the router.

Figure 18.6 shows the idea.

In this case, the forwarding decision is based on the value of the label, or virtual circuit identifier, as it is sometimes called.

To create a connection-oriented service, a three-phase process is used: setup, data transfer, and teardown.

In the setup phase, the source and destination addresses of the sender and receiver are used to make table entries for the connection-oriented service.

In the teardown phase, the source and destination inform the router to delete the corresponding entries.

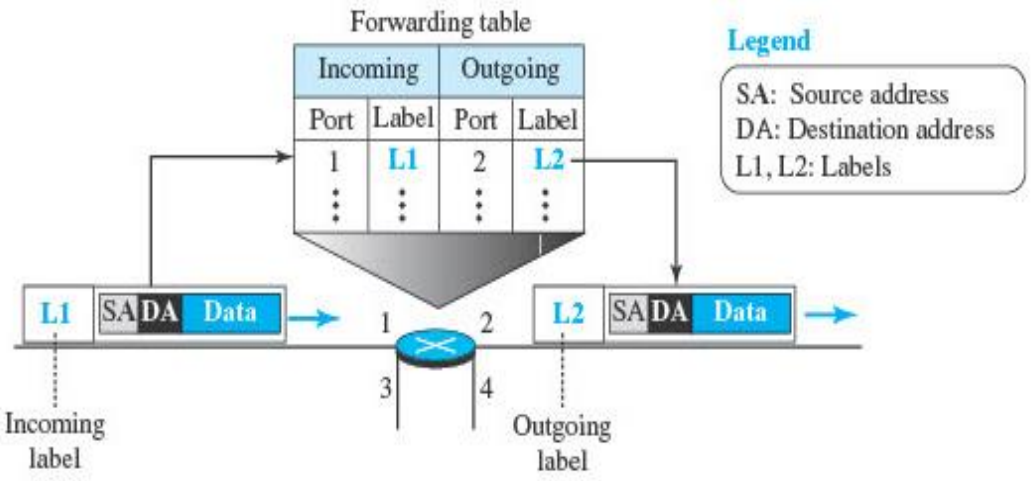
Data transfer occurs between these two phases.

In the setup phase, a router creates an entry for a virtual circuit.

For example, suppose source A needs to create a virtual circuit to destination B.

Two auxiliary packets need to be exchanged between the sender and the receiver: the request packet and the acknowledgment packet.

Figure 18.6 *Forwarding process in a router when used in a virtual-circuit network*



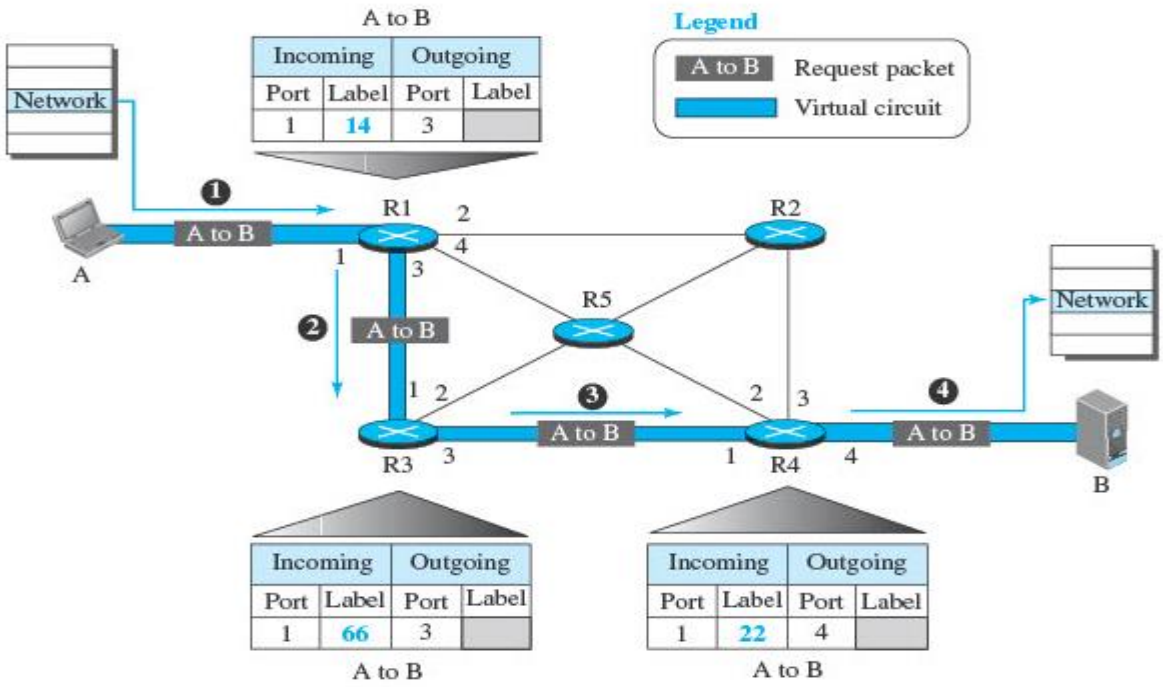
In the virtual-circuit approach, the forwarding decision is based on the label of the packet.

A request packet is sent from the source to the destination.

This auxiliary packet carries the source and destination addresses.

Figure 18.7 shows the process.

Figure 18.7 *Sending request packet in a virtual-circuit network*



1. Source A sends a request packet to router R1.

2. Router R1 receives the request packet. It knows that a packet going from A to B goes out through port 3. How the router has obtained this information is a point covered later. For the moment, assume that it knows the output port. The router creates an entry in its table for this virtual circuit, but it is only able to fill three of the four columns. The router assigns the incoming port (1) and chooses an available incoming label (14) and the outgoing port (3). It does not yet know the outgoing label, which will be found during the acknowledgment step. The router then forwards the packet through port 3 to router R3.

3. Router R3 receives the setup request packet. The same events happen here as at router R1; three columns of the table are completed: in this case, incoming port (1), incoming label (66), and outgoing port (3).

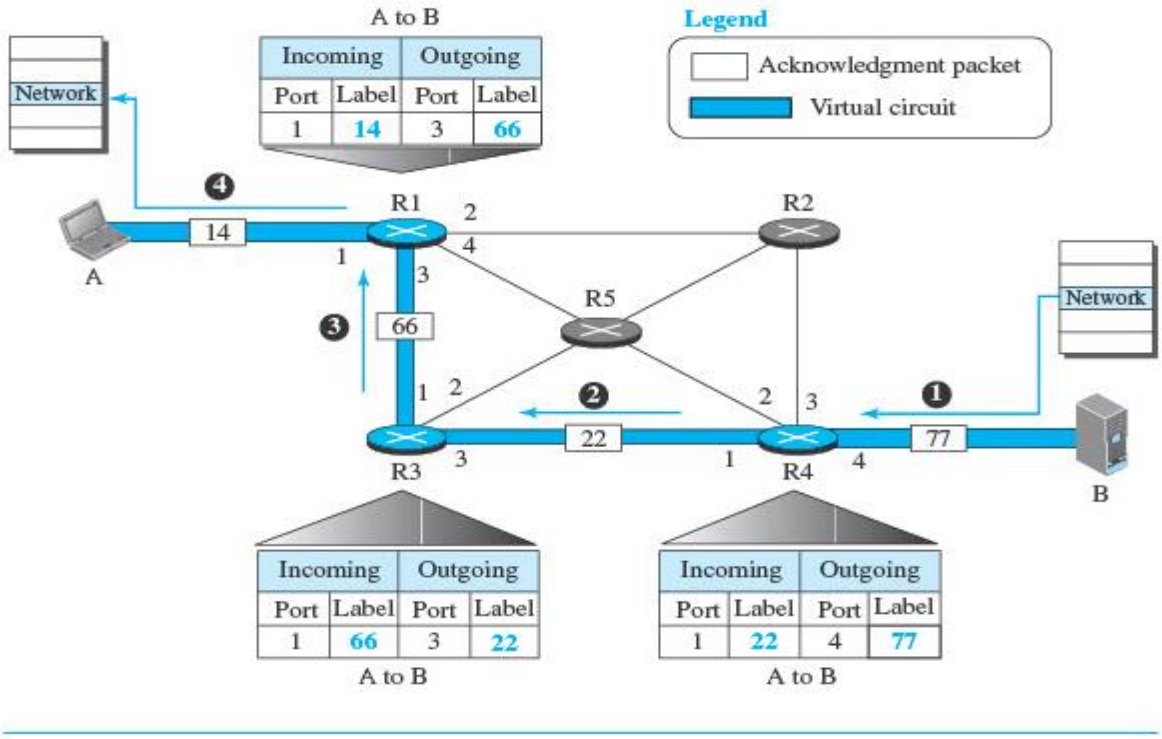
4. Router R4 receives the setup request packet. Again, three columns are completed: incoming port (1), incoming label (22), and outgoing port (4).

5. Destination B receives the setup packet, and if it is ready to receive packets from A, it assigns a label to the incoming packets that come from A, in this case 77, as shown in Figure 18.8. This label lets the destination know that the packets come from A, and not from other sources.

A special packet, called the acknowledgment packet, completes the entries in the switching tables.

Figure 18.8 shows the process.

Figure 18.8 *Sending acknowledgments in a virtual-circuit network*



1. The destination sends an acknowledgment to router R4. The acknowledgment carries the global source and destination addresses so the router knows which entry in the table is to be completed. The packet also carries label 77, chosen by the destination as the incoming label for packets from A. Router R4 uses this label to complete the outgoing label column for this entry. Note that 77 is the incoming label for destination B, but the outgoing label for router R4.

2. Router R4 sends an acknowledgment to router R3 that contains its incoming label in the table, chosen in the setup phase. Router R3 uses this as the outgoing label in the table.

3. Router R3 sends an acknowledgment to router R1 that contains its incoming label in the table, chosen in the setup phase. Router R1 uses this as the outgoing label in the table.

4. Finally router R1 sends an acknowledgment to source A that contains its incoming label in the table, chosen in the setup phase.

5. The source uses this as the outgoing label for the data packets to be sent to destination B.

The second phase is called the data-transfer phase.

After all routers have created their forwarding table for a specific virtual circuit, then the network-layer packets belonging to one message can be sent one after another.

In Figure 18.9, we show the flow of a single packet, but the process is the same for 1, 2, or 100 packets.

The source computer uses the label 14, which it has received from router R1 in the setup phase.

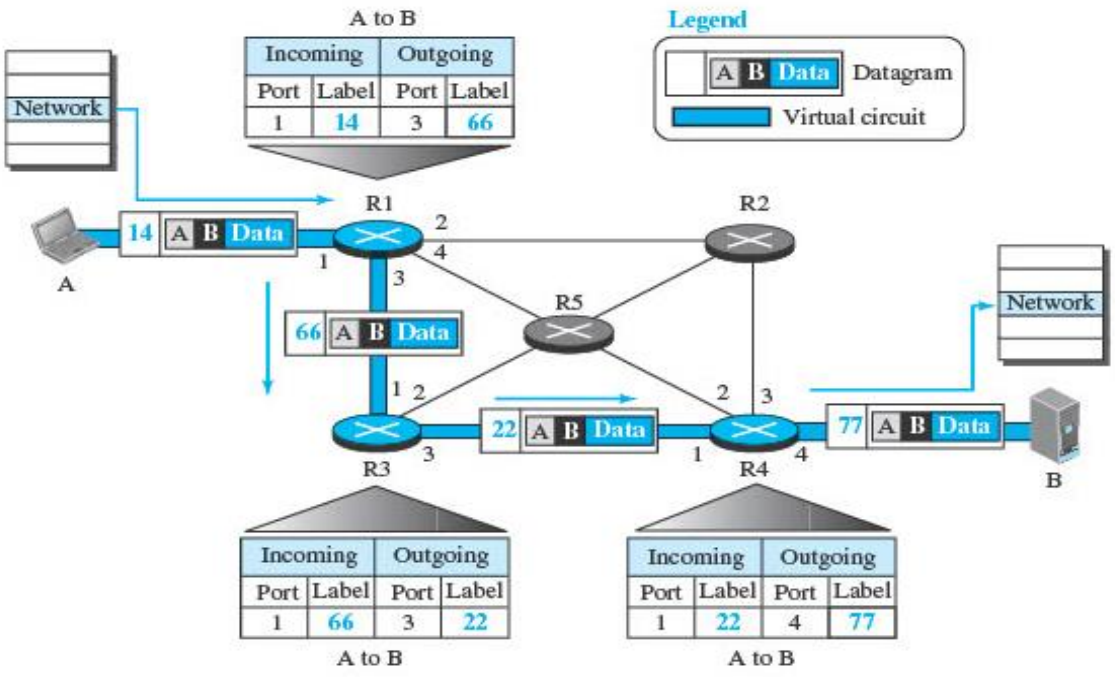
Router R1 forwards the packet to router R3, but changes the label to 66.

Router R3 forwards the packet to router R4, but changes the label to 22.

Finally, router R4 delivers the packet to its final destination with the label 77.

All the packets in the message follow the same sequence of labels, and the packets arrive in order at the destination.

Figure 18.9 *Flow of one packet in an established virtual circuit*



In the teardown phase, source A, after sending all packets to B, sends a special packet called a teardown packet.

Destination B responds with a confirmation packet.

All routers delete the corresponding entries from their tables.

Q6)a Solution :

In this section, we begin by discussing the first service provided by IPv4, packetizing.

We show how IPv4 defines the format of a packet in which the data coming from the upper layer or other protocols are encapsulated.

Packets used by the IP are called datagrams.

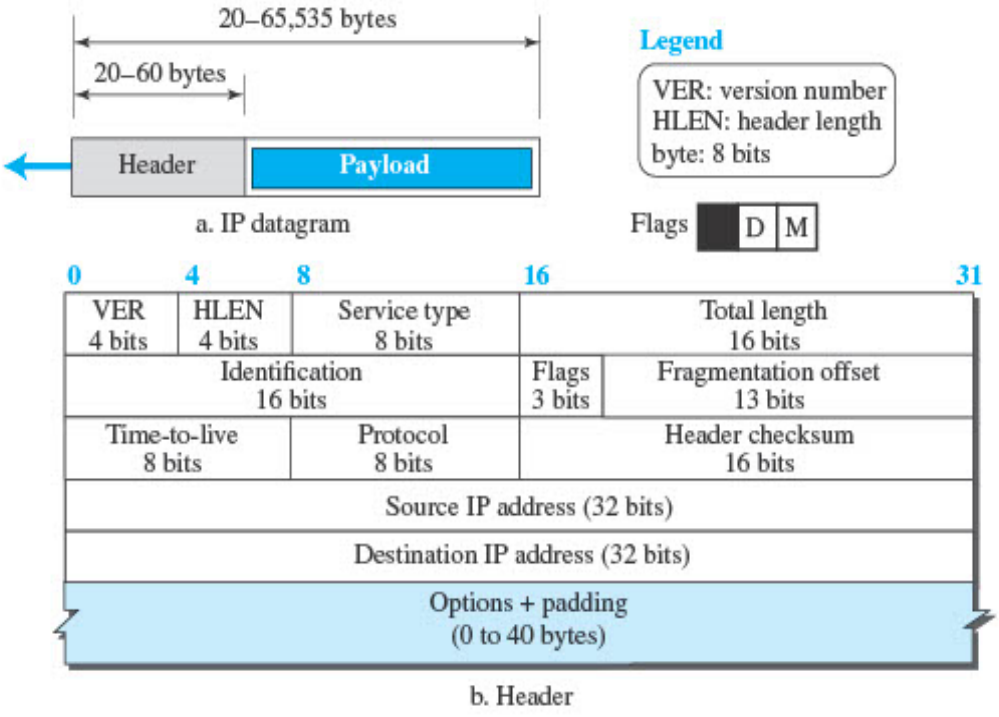
Figure 19.2 shows the IPv4 datagram format.

A datagram is a variable-length packet consisting of two parts: header and payload (data).

The header is 20 to 60 bytes in length and contains information essential to routing and delivery.

It is customary in TCP/IP to show the header in 4-byte sections.

Figure 19.2 *IP datagram*



Discussing the meaning and rationale for the existence of each field is essential to understanding the operation of IPv4; a brief description of each field is in order.

Version Number: The 4-bit version number (VER) field defines the version of the IPv4 protocol, which, obviously, has the value of 4.

Header Length: The 4-bit header length (HLEN) field defines the total length of the datagram header in 4-byte words. The IPv4 datagram has a variable-length header. When a device receives a datagram, it needs to know when the header stops and

the data, which is encapsulated in the packet, starts. However, to make the value of the header length (number of bytes) fit in a 4-bit header length, the total length of the header is calculated as 4-byte words. The total length is divided by 4 and the value is inserted in the field. The receiver needs to multiply the value of this field by 4 to find the total length.

Service Type: In the original design of the IP header, this field was referred to as type of service (TOS), which defined how the datagram should be handled. In the late 1990s, IETF redefined the field to provide differentiated services (DiffServ).

When we discuss differentiated services in Chapter 30, we will be in a better situation to define the bits in this field.

The use of 4-byte words for the length header is also logical because the IP header always needs to be aligned in 4-byte boundaries.

Total Length: This 16-bit field defines the total length (header plus data) of the IP datagram in bytes. A 16-bit number can define a total length of up to 65,535 (when all bits are 1s). However, the size of the datagram is normally much less than this. This field helps the receiving device to know when the packet has completely arrived. To find the length of the data coming from the upper layer, subtract the header length from the total length. The header length can be found by multiplying the value in the HLEN field by 4.

Length of data = total length − (HLEN) × 4

Though a size of 65,535 bytes might seem large, the size of the IPv4 datagram may increase in the near future as the underlying technologies allow even more throughput (greater bandwidth).

One may ask why we need this field anyway.

When a machine (router or host) receives a frame, it drops the header and the trailer, leaving the datagram.

Why include an extra field that is not needed?

The answer is that in many cases we really do not need the value in this field.

However, there are occasions in which the datagram is not the only thing encapsulated in a frame; it may be that padding has been added.

For example, the Ethernet protocol has a minimum and maximum restriction on the size of data that can be encapsulated in a frame (46 to 1500 bytes).

If the size of an IPv4 datagram is less than 46 bytes, some padding will be added to meet this requirement.

In this case, when a machine decapsulates the datagram, it needs to check the total length field to determine how much is really data and how much is padding.

Identification, Flags, and Fragmentation Offset: These three fields are related to the fragmentation of the IP datagram when the size of the datagram is larger than the underlying network can carry. We discuss the contents and importance of these fields when we talk about fragmentation in the next section.

Time-to-live: Due to some malfunctioning of routing protocols (discussed later) a datagram may be circulating in the Internet, visiting some networks over and over without reaching the destination. This may create extra traffic in the Internet. The time-to-live (TTL) field is used to control the maximum number of hops (routers) visited by the datagram. When a source host sends the datagram, it stores a number in this field. This value is approximately two times the maximum number of routers between any two hosts. Each router that processes the datagram decrements this number by one. If this value, after being decremented, is zero, the router discards the datagram.

Protocol: In TCP/IP, the data section of a packet, called the payload, carries the whole packet from another protocol. A datagram, for example, can carry a packet belonging to any transport-layer protocol such as UDP or TCP. A datagram can also carry a packet from other protocols that directly use the service of the IP, such as some routing protocols or some auxiliary protocols.

IP is not a reliable protocol; it does not check whether the payload carried by a datagram is corrupted during the transmission.

IP puts the burden of error checking of the payload on the protocol that owns the payload, such as UDP or TCP.

The datagram header, however, is added by IP, and its error-checking is the responsibility of IP.

Errors in the IP header can be a disaster.

For example, if the destination IP address is corrupted, the packet can be delivered to the wrong host.

If the protocol field is corrupted, the payload may be delivered to the wrong protocol.

If the fields related to the fragmentation are corrupted, the datagram cannot be reassembled correctly at the destination, and so on.

For these reasons, IP adds a header checksum field to check the header, but not the payload.

We need to remember that, since the value of some fields, such as TTL, which are related to fragmentation and options, may change from router to router, the checksum needs to be recalculated at each router.

As we discussed in Chapter 10, checksum in the Internet normally uses a 16-bit field, which is the complement of the sum of other fields calculated using 1s complement arithmetic.

These 32-bit source and destination address fields define the IP address of the source and destination respectively.

The source host should know its IP address.

The destination IP address is either known by the protocol that uses the service of IP or is provided by the DNS as described in Chapter 26.

Note that the value of these fields must remain unchanged during the time the IP datagram travels from the source host to the destination host.

IP addresses were discussed in Chapter 18.

A datagram header can have up to 40 bytes of options.

Options can be used for network testing and debugging.

Although options are not a required part of the IP header, option processing is required of the IP software.

This means that all implementations must be able to handle options if they are present in the header.

The existence of options in a header creates some burden on the datagram handling; some options can be changed by routers, which forces each router to recalculate the header checksum.

There are one-byte and multi-byte options that we will briefly discuss later in the chapter.

Payload, or data, is the main reason for creating a datagram.

Payload is the packet coming from other protocols that use the service of IP.

Comparing a datagram to a postal package, payload is the content of the package; the header is only the information written on the package.

Q6)b Solution :

The distance-vector (DV) routing uses the goal we discussed in the introduction, to find the best route.

In distance-vector routing, the first thing each node creates is its own least-cost tree with the rudimentary information it has about its immediate neighbors.

The incomplete trees are exchanged between immediate neighbors to make the trees more and more complete and to represent the whole internet.

We can say that in distance-vector routing, a router continuously tells all of its neighbors what it knows about the whole internet (although the knowledge can be incomplete).

Before we show how incomplete least-cost trees can be combined to make complete ones, we need to discuss two important topics: the Bellman-Ford equation and the concept of distance vectors, which we cover next.

The heart of distance-vector routing is the famous Bellman-Ford equation.

This equation is used to find the least cost (shortest distance) between a source node, x, and a destination node, y, through some intermediary nodes (a, b, c, . . .) when the costs between the source and the intermediary nodes and the least costs between the intermediary nodes and the destination are given.

The following shows the general case in which Dij is the shortest distance and cij is the cost between nodes i and j.

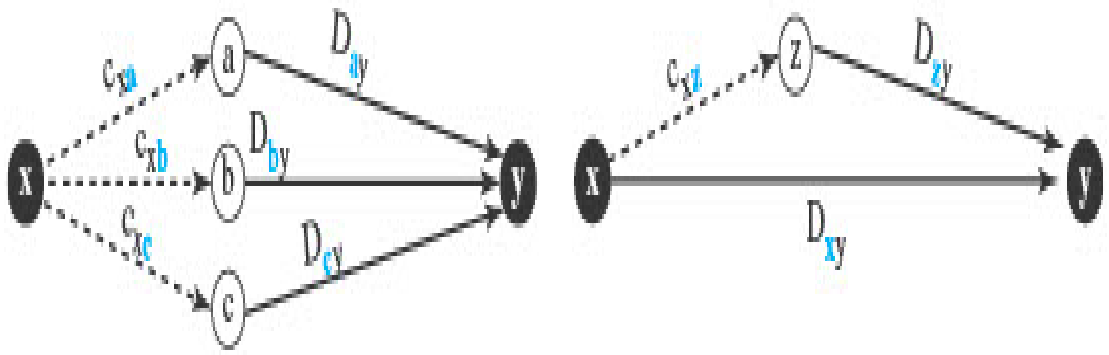
$$D_{xy} = \min\left\{(c_{xa} + D_{ay}), (c_{xb} + D_{by}), (c_{xc} + D_{cy}), \ldots\right\}$$

In distance-vector routing, normally we want to update an existing least cost with a least cost through an intermediary node, such as z, if the latter is shorter. In this case, the equation becomes simpler, as shown below:

$$D_{xy} = \min\left\{D_{xy}, (c_{xz} + D_{zy})\right\}$$

Figure 20.3 shows the idea graphically for both cases.

Figure 20.3 *Graphical idea behind Bellman-Ford equation*



a. General case with three intermediate nodes

b. Updating a path with a new route

We can say that the Bellman-Ford equation enables us to build a new least-cost path from previously established least-cost paths.

In Figure 20.3, we can think of (a→y), (b→y), and (c→y) as previously established least-cost paths and (x→y) as the new least-cost path.

We can even think of this equation as the builder of a new least-cost tree from previously established least-cost trees if we use the equation repeatedly.

In other words, the use of this equation in distance-vector routing is a witness that this method also uses least-cost trees, but this use may be in the background.

We will shortly show how we use the Bellman-Ford equation and the concept of distance vectors to build least-cost paths for each node in distance-vector routing, but first we need to discuss the concept of a distance vector.

Distance-Vector Routing Algorithm

Now we can give a simplified pseudocode for the distance-vector routing algorithm, as shown in Table 20.1.

The algorithm is run by its node independently and asynchronously.

Table 20.1 *Distance-Vector Routing Algorithm for a Node*

```
1  Distance_Vector_Routing ( )
2  {
3      // Initialize (create initial vectors for the node)
4      D[myself] = 0
```

Table 20.1 *Distance-Vector Routing Algorithm for a Node (continued)*

```
5     for (y = 1 to N)
6     {
7         if (y is a neighbor)
8             D[y] = c[myself][y]
9         else
10            D[y] = ∞
11    }
12    send vector {D[1], D[2], ..., D[N]} to all neighbors
13    // Update (improve the vector with the vector received from a neighbor)
14    repeat (forever)
15    {
16        wait (for a vector Dw from a neighbor w or any change in the link)
17        for (y = 1 to N)
18        {
19            D[y] = min [D[y], (c[myself][w] + Dw[y])]      // Bellman-Ford equation
20        }
21        if (any change in the vector)
22            send vector {D[1], D[2], ..., D[N]} to all neighbors
23    }
24  } // End of Distance Vector
```

,

Lines 4 to 11 initialize the vector for the node.

Lines 14 to 23 show how the vector can be updated after receiving a vector from the immediate neighbor.

The for loop in lines 17 to 20 allows all entries (cells) in the vector to be updated after receiving a new vector.

Note that the node sends its vector in line 12, after being initialized, and in line 22, after it is updated.

Program :

```
#include <stdio.h>

int main()
{
    int next_hop[20][20];
    int cost[20][20], n;
```

```
int i,j,k;

printf( "Enter no of nodes.\n" );
scanf("%d",&n);
printf( "Enter the distance(cost or weight)matrix:\n");
printf("enter zero for same node connection and 100 for no direct connection\n");
for ( i = 1; i <=n; i++)
  printf("\t%d",i); // print column nos.
for ( i = 1; i <= n; i++)
{
        printf("\n%d\t",i); // print row nos.
      for ( j = 1; j <= n; j++)
      {
         scanf("%d",&cost[i][j]); // accept cost matrix from user

          next_hop[i][j]=j; // initialize the next hop matrix
      }
}
      for ( k = 1; k <= n; k++) {
          for ( i = 1; i <=n; i++) {
                for ( j = 1; j <=n; j++) {

                    if(cost[i][j]>cost[i][k]+cost[k][j])
                    {
                    cost[i][j]= (cost[i][k]+cost[k][j]); // update cost matrix
                    next_hop[i][j]=k; // update next hop matrix
                    }
                }
          }
```

```
        }
    //print cost matrix
        printf(" cost matrix: \n");
        for ( i = 1; i <=n; i++)
      printf("\t%d",i);
        for(i=1;i<=n;i++)
        {
        printf("\n%d\t",i);
        for ( j = 1; j <= n; j++)
        {
           printf("%d\t",cost[i][j]);

        }
    printf("\n");
    }
//print next hop matrix
        printf(" Next hop matrix: \n");
        for ( i = 1; i <=n; i++)
         printf("\t%d",i);
        for(i=1;i<=n;i++)
        {
        printf("\n%d\t",i);
        for ( j = 1; j <= n; j++)
        {
           printf("%d\t",next_hop[i][j]);

        }
    printf("\n");
    }
```

```
    for ( i = 1; i <=n; i++)
          {
          printf( "Routing table info for router:%d \n", i );
           printf("Dest\tNext Hop\tDist\n" );
          for ( j = 1; j <= n; j++)
               printf("%d\t%d\t\t%d\n", j,next_hop[i][j], cost[i][j]);
    }



}
```

Q7)a Solution :

A transport-layer protocol, like a network-layer protocol, can provide two types of services: connectionless and connection-oriented.

The nature of these services at the transport layer, however, is different from the ones at the network layer.

At the network layer, a connectionless service may mean different paths for different datagrams belonging to the same message.

At the transport layer, we are not concerned about the physical paths of packets (we assume a logical connection between two transport layers).

Connectionless service at the transport layer means independency between packets; connection-oriented means dependency.

Let us elaborate on these two services.

Connectionless Service

In a connectionless service, the source process (application program) needs to divide its message into chunks of data of the size acceptable by the transport layer and deliver them to the transport layer one by one.

The transport layer treats each chunk as a single unit without any relation between the chunks.

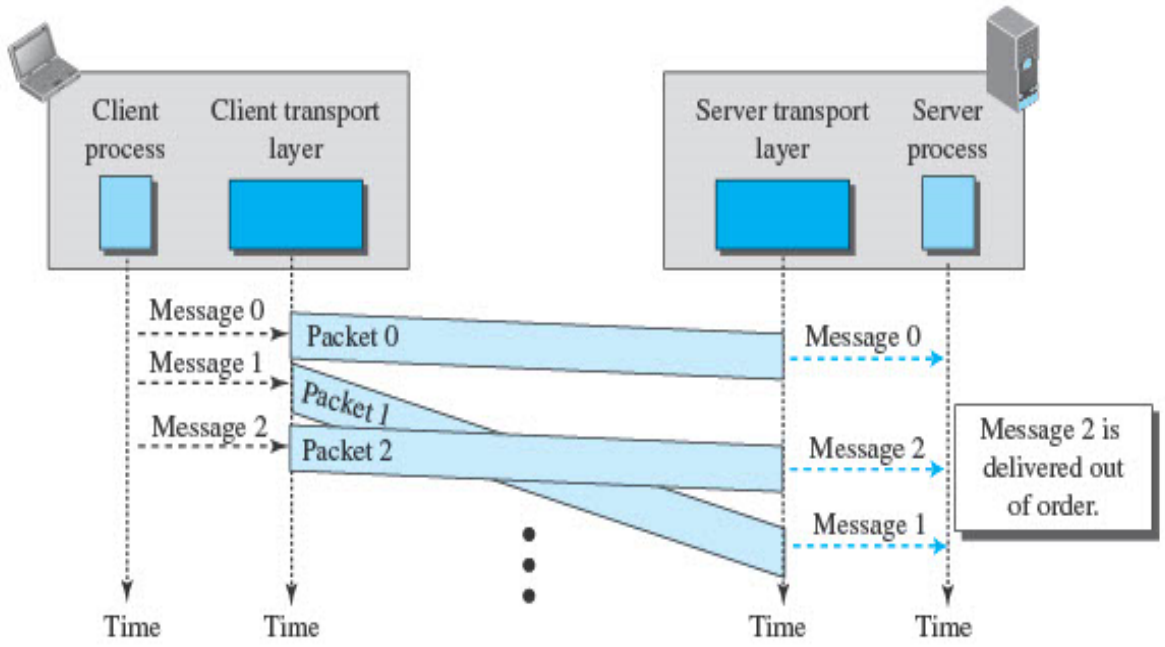
When a chunk arrives from the application layer, the transport layer encapsulates it in a packet and sends it.

To show the independency of packets, assume that a client process has three chunks of messages to send to a server process.

The chunks are handed over to the connectionless transport protocol in order.

However, since there is no dependency between the packets at the transport layer, the packets may arrive out of order at the destination and will be delivered out of order to the server process (Figure 23.14).

Figure 23.14 *Connectionless service*



In Figure 23.14, we have shown the movement of packets using a time line, but we have assumed that the delivery of the process to the transport layer and vice versa are instantaneous.

The figure shows that at the client site, the three chunks of messages are delivered to the client transport layer in order (0, 1, and 2).

Because of the extra delay in transportation of the second packet, the delivery of messages at the server is not in order (0, 2, 1).

If these three chunks of data belong to the same message, the server process may have received a strange message.

The situation would be worse if one of the packets were lost.

Since there is no numbering on the packets, the receiving transport layer has no idea that one of the messages has been lost.

It just delivers two chunks of data to the server process.

The above two problems arise from the fact that the two transport layers do not coordinate with each other.

The receiving transport layer does not know when the first packet will come nor when all of the packets have arrived.

We can say that no flow control, error control, or congestion control can be effectively implemented in a connectionless service.

Connection-Oriented Service

In a connection-oriented service, the client and the server first need to establish a logical connection between themselves.

The data exchange can only happen after the connection establishment.

After data exchange, the connection needs to be torn down (Figure 23.15).

As we mentioned before, the connection-oriented service at the transport layer is different from the same service at the network layer.

In the network layer, connection-oriented service means a coordination between the two end hosts and all the routers in between.

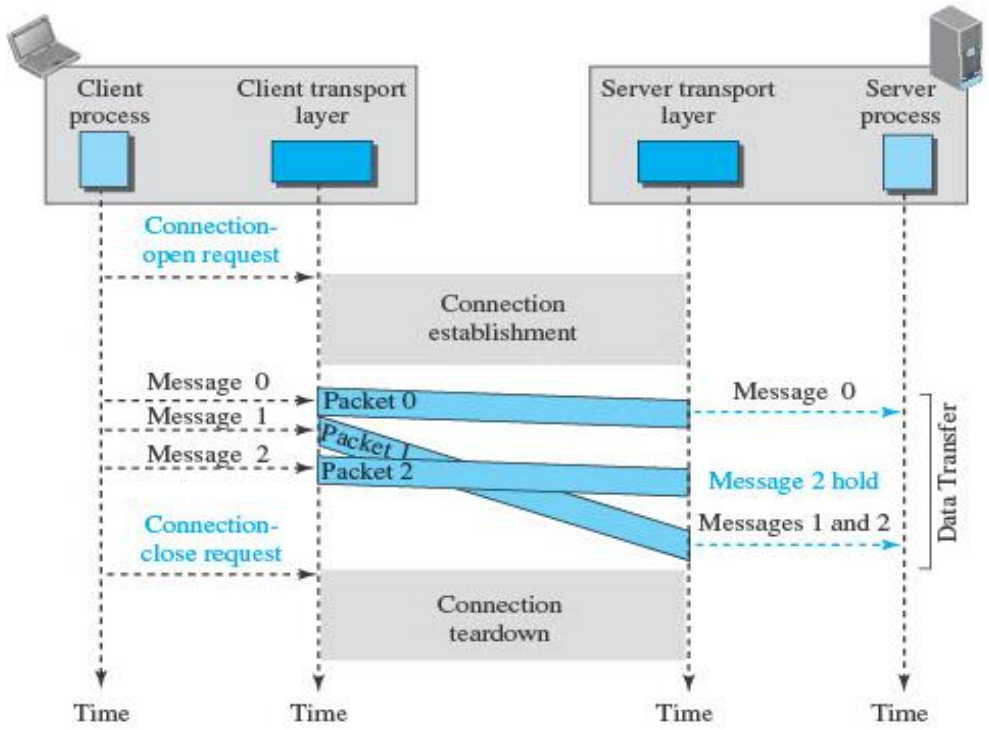
At the transport layer, connection-oriented service involves only the two hosts; the service is end to end.

This means that we should be able to make a connection-oriented protocol at the transport layer over either a connectionless or connection-oriented protocol at the network layer.

Figure 23.15 shows the connection-establishment, data-transfer, and tear-down phases in a connection-oriented service at the transport layer.

We can implement flow control, error control, and congestion control in a connection-oriented protocol.

Figure 23.15 *Connection-oriented service*



Q7)b Solution :

To improve the efficiency of transmission (to fill the pipe), multiple packets must be in transition while the sender is waiting for acknowledgment.

In other words, we need to let more than one packet be outstanding to keep the channel busy while the sender is waiting for acknowledgment.

In this section, we discuss one protocol that can achieve this goal; in the next section, we discuss a second.

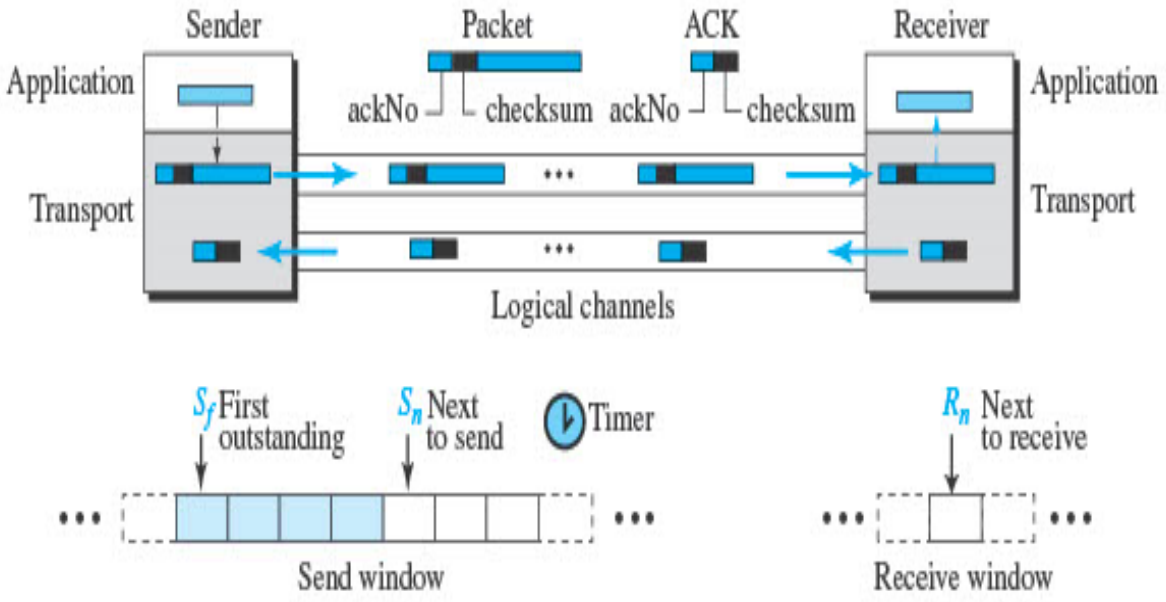
The first is called Go-Back-N (GBN) (the rationale for the name will become clear later).

The key to Go-back-N is that we can send several packets before receiving acknowledgments, but the receiver can only buffer one packet.

We keep a copy of the sent packets until the acknowledgments arrive.

Figure 23.23 shows the outline of the protocol. Note that several data packets and acknowledgments can be in the channel at the same time.

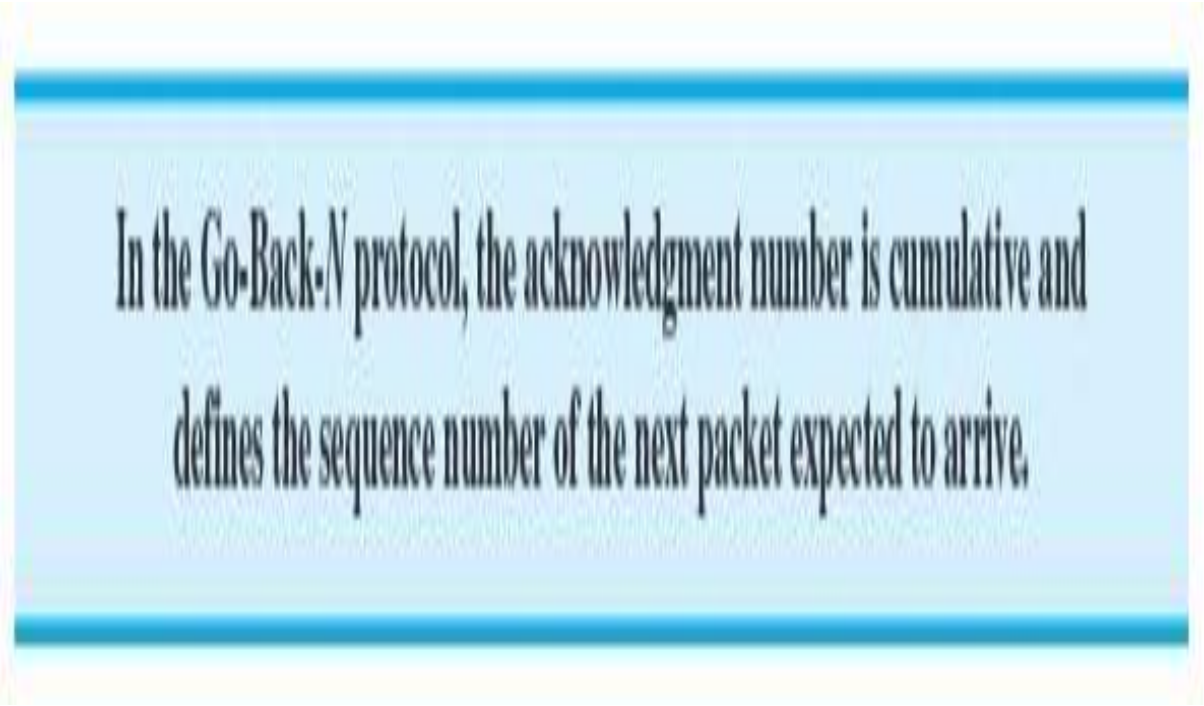
Figure 23.23 *Go-Back-N protocol*



As we mentioned before, the sequence numbers are modulo 2m, where m is the size of the sequence number field in bits.

An acknowledgment number in this protocol is cumulative and defines the sequence number of the next packet expected.

For example, if the acknowledgment number (ackNo) is 7, it means all packets with sequence number up to 6 have arrived, safe and sound, and the receiver is expecting the packet with sequence number 7.



The send window is an imaginary box covering the sequence numbers of the data packets that can be in transit or can be sent.

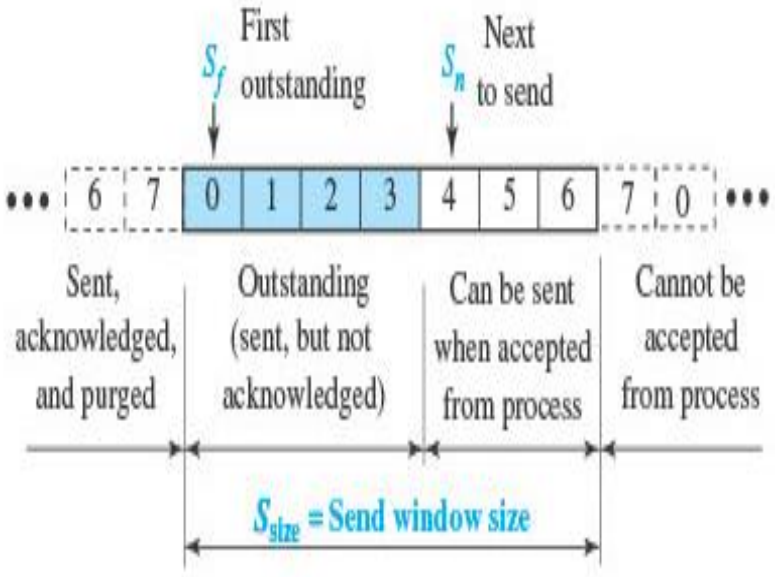
In each window position, some of these sequence numbers define the packets that have been sent; others define those that can be sent.

The maximum size of the window is 2m − 1, for reasons that we discuss later.

In this chapter, we let the size be fixed and set to the maximum value, but we will see later that some protocols may have a variable window size.

Figure 23.24 shows a sliding window of size 7 (m = 3) for the Go-Back-N protocol.

Figure 23.24 *Send window for Go-Back-N*



The send window at any time divides the possible sequence numbers into four regions.

The first region, left of the window, defines the sequence numbers belonging to packets that are already acknowledged.

The sender does not worry about these packets and keeps no copies of them.

The second region, colored, defines the range of sequence numbers belonging to the packets that have been sent, but have an unknown status.

The sender needs to wait to find out if these packets have been received or were lost.

We call these outstanding packets.

The third range, white in the figure, defines the range of sequence numbers for packets that can be sent; however, the corresponding data have not yet been received from the application layer.

Finally, the fourth region, right of the window, defines sequence numbers that cannot be used until the window slides.

The window itself is an abstraction; three variables define its size and location at any time.

We call these variables Sf (send window, the first outstanding packet), Sn (send window, the next packet to be sent), and Ssize (send window, size).

The variable Sf defines the sequence number of the first (oldest) outstanding packet.

The variable Sn holds the sequence number that will be assigned to the next packet to be sent.

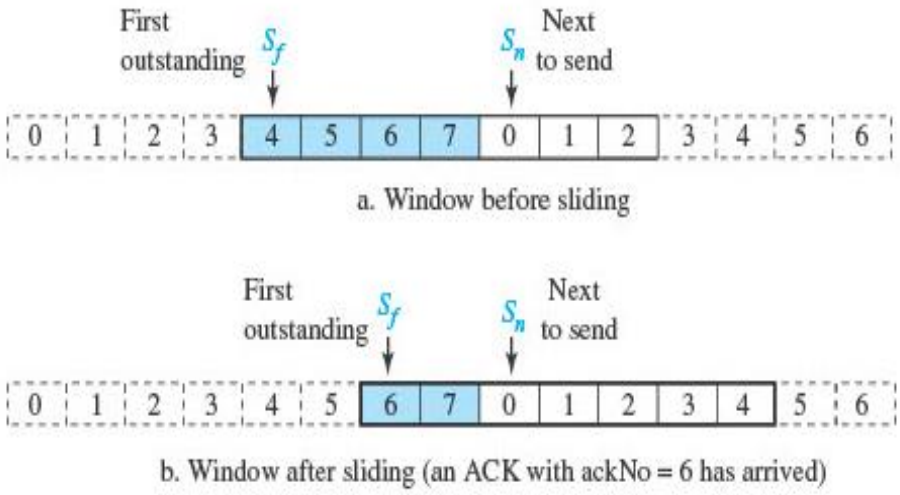
Finally, the variable Ssize defines the size of the window, which is fixed in our protocol.

Figure 23.25 shows how a send window can slide one or more slots to the right when an acknowledgment arrives from the other end.

In the figure, an acknowledgment with ackNo = 6 has arrived. This means that the receiver is waiting for packets with sequence number 6.

The send window is an abstract concept defining an imaginary box of maximum size = $2^m - 1$ with three variables: S_f, S_n, and S_{size}.

Figure 23.25 *Sliding the send window*



The send window can slide one or more slots when an error-free ACK with ackNo greater than or equal to S_f and less than S_n (in modular arithmetic) arrives.

The receive window makes sure that the correct data packets are received and that the correct acknowledgments are sent.

In Go-Back-N, the size of the receive window is always 1.

The receiver is always looking for the arrival of a specific packet.

Any packet arriving out of order is discarded and needs to be resent.

Figure 23.26 shows the receive window.

Note that we need only one variable, Rn (receive window, next packet expected), to define this abstraction.

The sequence numbers to the left of the window belong to the packets already received and acknowledged; the sequence numbers to the right of this window define the packets that cannot be received.

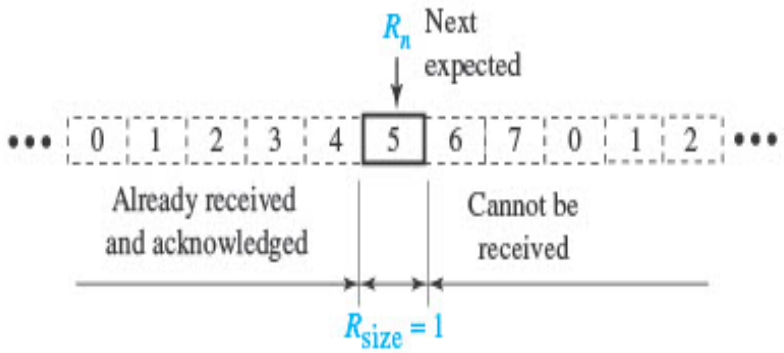
Any received packet with a sequence number in these two regions is discarded.

Only a packet with a sequence number matching the value of Rn is accepted and acknowledged.

The receive window also slides, but only one slot at a time.

When a correct packet is received, the window slides, Rn = (Rn + 1) modulo 2m.

Figure 23.26 *Receive window for Go-Back-N*

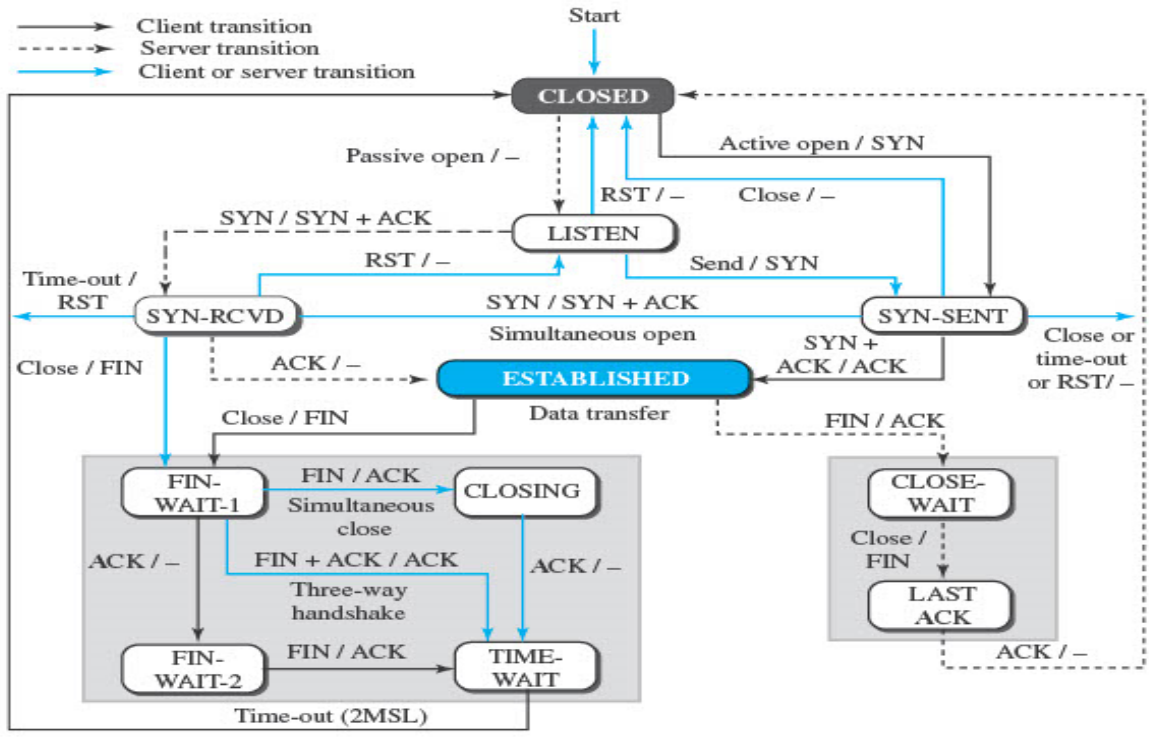


The receive window is an abstract concept defining an imaginary box of size 1 with a single variable R_n. The window slides when a correct packet has arrived; sliding occurs one slot at a time.

Q8)a Solution :

To keep track of all the different events happening during connection establishment, connection termination, and data transfer, TCP is specified as the finite state machine (FSM) as shown in Figure 24.14.

Figure 24.14 *State transition diagram*



The state marked *ESTABLISHED* in the FSM is in fact two different sets of states that the client and server undergo to transfer data.

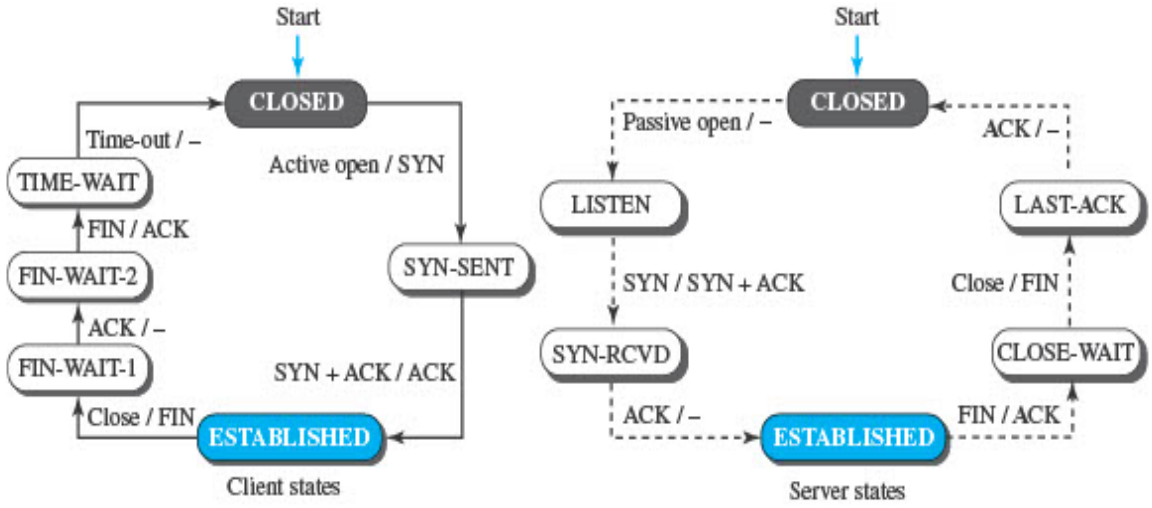
Table 24.2 *States for TCP*

State	*Description*
CLOSED	No connection exists
LISTEN	Passive open received; waiting for SYN
SYN-SENT	SYN sent; waiting for ACK
SYN-RCVD	SYN + ACK sent; waiting for ACK
ESTABLISHED	Connection established; data transfer in progress
FIN-WAIT-1	First FIN sent; waiting for ACK
FIN-WAIT-2	ACK to first FIN received; waiting for second FIN
CLOSE-WAIT	First FIN received, ACK sent; waiting for application to close
TIME-WAIT	Second FIN received, ACK sent; waiting for 2MSL time-out
LAST-ACK	Second FIN sent; waiting for ACK
CLOSING	Both sides decided to close simultaneously

A Half-Close Scenario

Figure 24.15 shows the state transition diagram for this scenario.

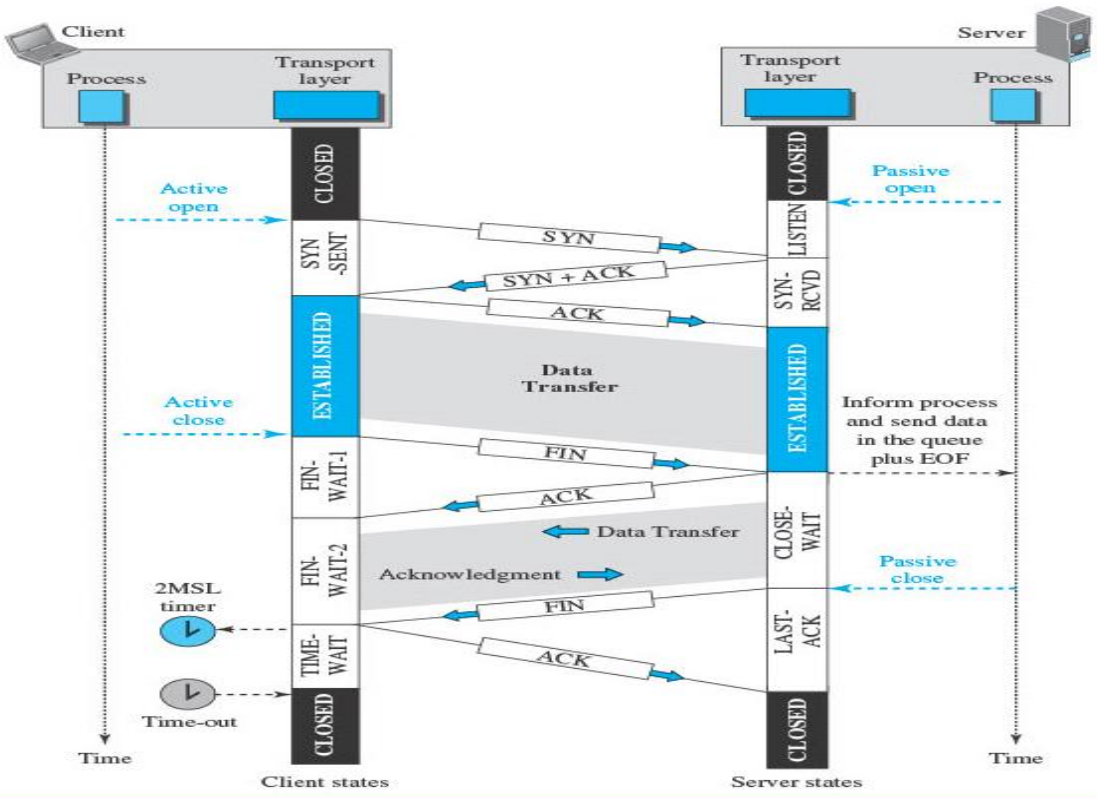
Figure 24.15 *Transition diagram with half-close connection termination*



The client process issues an *active open* command to its TCP to request a connection to a specific socket address. TCP sends a SYN segment and moves to the **SYN-SENT** state. After receiving the SYN + ACK segment, TCP sends an ACK segment and goes to the **ESTABLISHED** state. Data are transferred, possibly in both directions, and acknowledged. When the client process has no more data to send, it issues a command called an *active close*. The TCP sends a FIN segment and goes to the **FIN-WAIT-1** state. When it receives the ACK segment, it goes to the **FIN-WAIT-2** state. When the client receives a FIN segment, it sends an ACK segment and goes to the **TIME-WAIT** state. The client remains in this state for 2 MSL seconds (see TCP timers later in the chapter). When the corresponding timer expires, the client goes to the **CLOSED** state.

The server process issues a *passive open* command. The server TCP goes to the **LISTEN** state and remains there passively until it receives a SYN segment. The TCP then sends a SYN + ACK segment and goes to the **SYN-RCVD** state, waiting for the client to send an ACK segment. After receiving the ACK segment, TCP goes to the **ESTABLISHED** state, where data transfer can take place. TCP remains in this state until it receives a FIN segment from the client signifying that there are no more data to be exchanged and the connection can be closed. The server, upon receiving the FIN segment, sends all queued data to the server with a virtual EOF marker, which means that the connection must be closed. It sends an ACK segment and goes to the **CLOSE-WAIT** state, but postpones acknowledging the FIN segment received from the client until it receives a *passive close* command from its process. After receiving the passive close command, the server sends a FIN segment to the client and goes to the **LAST-ACK** state, waiting for the final ACK. When the ACK segment is received from the client, the server goes to the **CLOSE** state. Figure 24.16 shows the same scenario with states over the time line.

Figure 24.16 *Time-line diagram for a common scenario*



Q8)b Solution :

Earlier we discussed the general services provided by a transport-layer protocol.

In this section, we discuss what portions of those general services are provided by UDP.

UDP provides process-to-process communication using socket addresses, a combination of IP addresses and port numbers.

As mentioned previously, UDP provides a connectionless service.

This means that each user datagram sent by UDP is an independent datagram.

There is no relationship between the different user datagrams even if they are coming from the same source process and going to the same destination program.

The user datagrams are not numbered.

Also, unlike TCP, there is no connection establishment and no connection termination.

This means that each user datagram can travel on a different path.

One of the ramifications of being connectionless is that the process that uses UDP cannot send a stream of data to UDP and expect UDP to chop them into different, related user datagrams.

Instead each request must be small enough to fit into one user datagram.

Only those processes sending short messages, messages less than 65,507 bytes (65,535 minus 8 bytes for the UDP header and minus 20 bytes for the IP header), can use UDP.

UDP is a very simple protocol.

There is no flow control, and hence no window mechanism.

The receiver may overflow with incoming messages.

The lack of flow control means that the process using UDP should provide for this service, if needed.

There is no error control mechanism in UDP except for the checksum.

This means that the sender does not know if a message has been lost or duplicated.

When the receiver detects an error through the checksum, the user datagram is silently discarded.

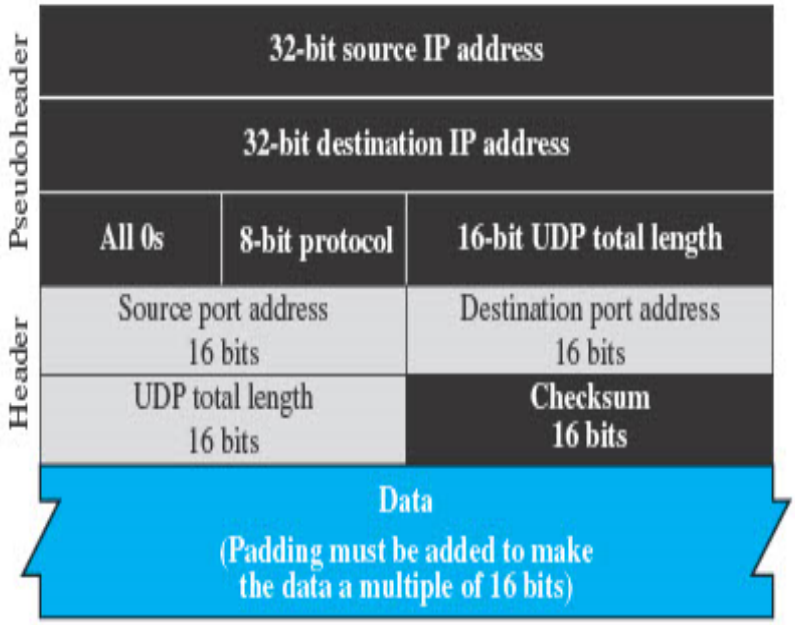
The lack of error control means that the process using UDP should provide for this service, if needed.

We discussed checksum and its calculation in Chapter 10.

UDP checksum calculation includes three sections: a pseudoheader, the UDP header, and the data coming from the application layer.

The pseudoheader is the part of the header of the IP packet (discussed in Chapter 19) in which the user datagram is to be encapsulated with some fields filled with 0s (see Figure 24.3).

Figure 24.3 *Pseudoheader for checksum calculation*



If the checksum does not include the pseudoheader, a user datagram may arrive safe and sound.

However, if the IP header is corrupted, it may be delivered to the wrong host.

The protocol field is added to ensure that the packet belongs to UDP, and not to TCP.

We will see later that if a process can use either UDP or TCP, the destination port number can be the same.

The value of the protocol field for UDP is 17.

If this value is changed during transmission, the checksum calculation at the receiver will detect it and UDP drops the packet.

It is not delivered to the wrong protocol.

The sender of a UDP packet can choose not to calculate the checksum.

In this case, the checksum field is filled with all 0s before being sent.

In the situation where the sender decides to calculate the checksum, but it happens that the result is all 0s, the checksum is changed to all 1s before the packet is sent.

In other words, the sender complements the sum two times.

Note that this does not create confusion because the value of the checksum is never all 1s in a normal situation.

Since UDP is a connectionless protocol, it does not provide congestion control.

UDP assumes that the packets sent are small and sporadic and cannot create congestion in the network.

This assumption may or may not be true today, when UDP is used for interactive real-time transfer of audio and video.

To send a message from one process to another, the UDP protocol encapsulates and decapsulates messages.

We have talked about ports without discussing the actual implementation of them.

In UDP, queues are associated with ports.

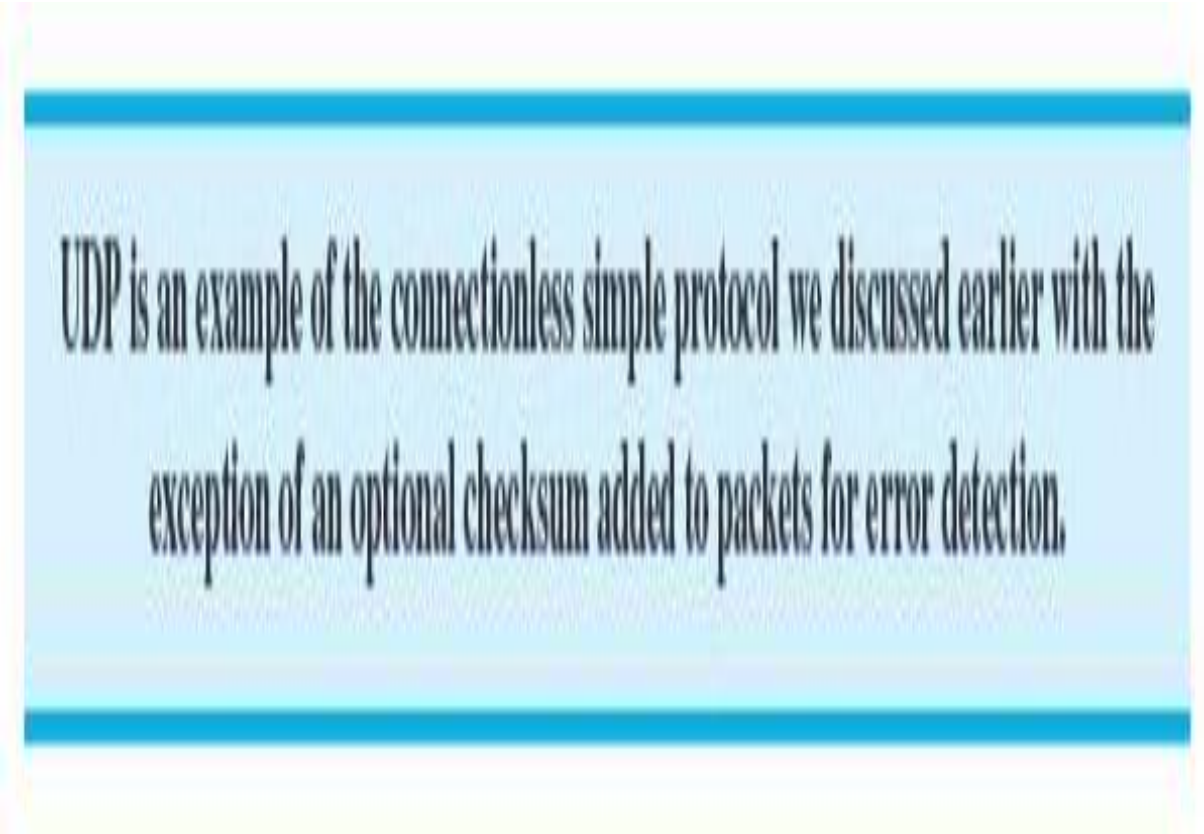
At the client site, when a process starts, it requests a port number from the operating system.

Some implementations create both an incoming and an outgoing queue associated with each process.

Other implementations create only an incoming queue associated with each process.

In a host running a TCP/IP protocol suite, there is only one UDP but possibly several processes that may want to use the services of UDP.

To handle this situation, UDP multiplexes and demultiplexes.



Q9)a Solution :

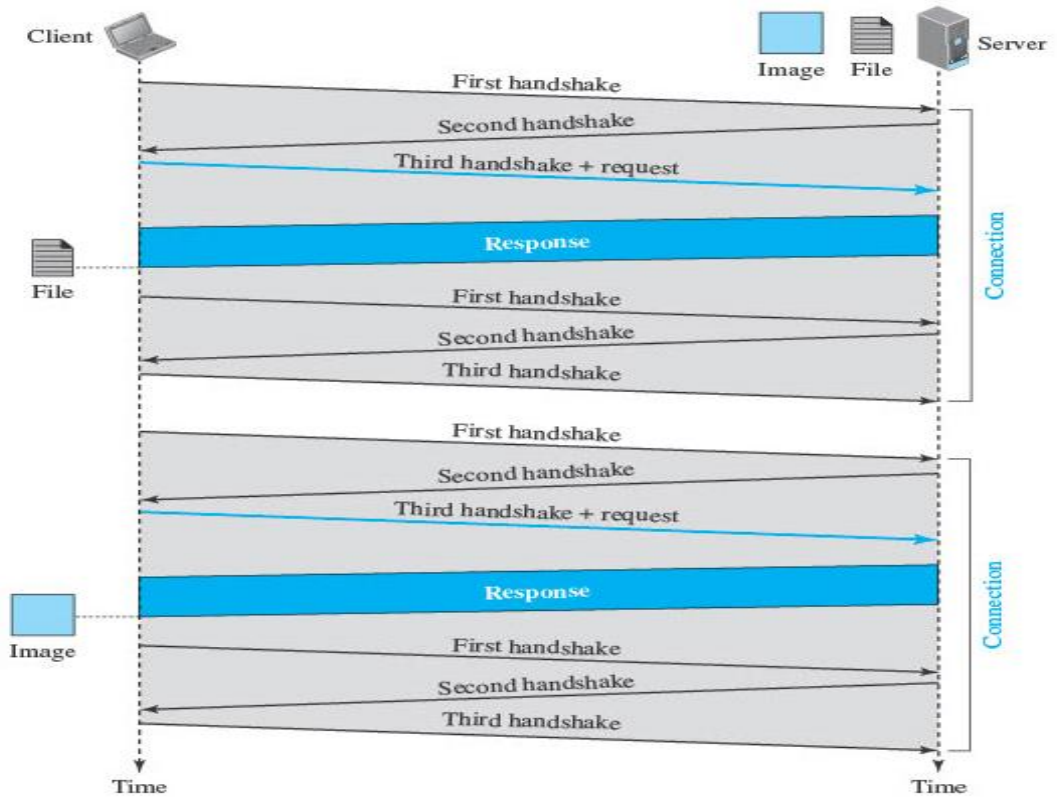
i) WWW (World Wide Web) [Out of syllabus].
ii) HTTP :

The HyperText Transfer Protocol (HTTP) is used to define how the client-server programs can be written to retrieve web pages from the Web.
An HTTP client sends a request; an HTTP server returns a response.
The server uses the port number 80; the client uses a temporary port number.
HTTP uses the services of TCP, which, as discussed before, is a connection-oriented and reliable protocol.
This means that, before any transaction between the client and the server can take place, a connection needs to be established between them.
After the transaction, the connection should be terminated.
The client and server, however, do not need to worry about errors in messages exchanged or loss of any message, because the TCP is reliable and will take care of this matter, as we saw in Chapter 24.

As we discussed in the previous section, the hypertext concept embedded in web page documents may require several requests and responses.
If the web pages, objects to be retrieved, are located on different servers, we do not have any other choice than to create a new TCP connection for retrieving each object.
However, if some of the objects are located on the same server, we have two choices: to retrieve each object using a new TCP connection or to make a TCP connection and retrieve them all.
The first method is referred to as a nonpersistent connection, the second as a persistent connection.
HTTP, prior to version 1.1, specified nonpersistent connections, while persistent connections are the default in version 1.1, but it can be changed by the user.

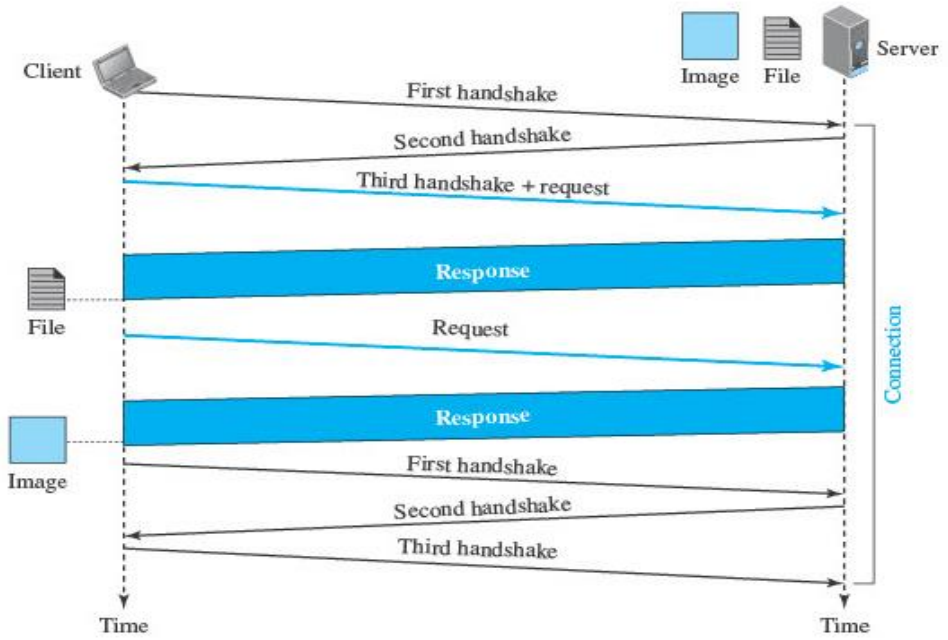
In a nonpersistent connection, one TCP connection is made for each request/response.
The following lists the steps in this strategy:
1. The client opens a TCP connection and sends a request.
2. The server sends the response and closes the connection.
3. The client reads the data until it encounters an end-of-file marker; it then closes the connection.
In this strategy, if a file contains links to N different pictures in different files (all located on the same server), the connection must be opened and closed N + 1 times.
The nonpersistent strategy imposes high overhead on the server because the server needs N + 1 different buffers each time a connection is opened.

Figure 26.3 *Example 26.3*



HTTP version 1.1 specifies a persistent connection by default.
In a persistent connection, the server leaves the connection open for more requests after sending a response.
The server can close the connection at the request of a client or if a time-out has been reached.
The sender usually sends the length of the data with each response.
However, there are some occasions when the sender does not know the length of the data.
This is the case when a document is created dynamically or actively.
In these cases, the server informs the client that the length is not known and closes the connection after sending the data so the client knows that the end of the data has been reached.
Time and resources are saved using persistent connections.
Only one set of buffers and variables needs to be set for the connection at each site.
The round trip time for connection establishment and connection termination is saved.

Figure 26.4 *Example 26.4*



iii) FTP :

File Transfer Protocol (FTP) is the standard protocol provided by TCP/IP for copying a file from one host to another.

Although transferring files from one system to another seems simple and straightforward, some problems must be dealt with first.

For example, two systems may use different file name conventions.

Two systems may have different ways to represent data.

Two systems may have different directory structures.

All of these problems have been solved by FTP in a very simple and elegant approach.

Although we can transfer files using HTTP, FTP is a better choice to transfer large files or to transfer files using different formats.

Figure 26.10 shows the basic model of FTP.

The client has three components: the user interface, the client control process, and the client data transfer process.

The server has two components: the server control process and the server data transfer process.

The control connection is made between the control processes.

The data connection is made between the data transfer processes.

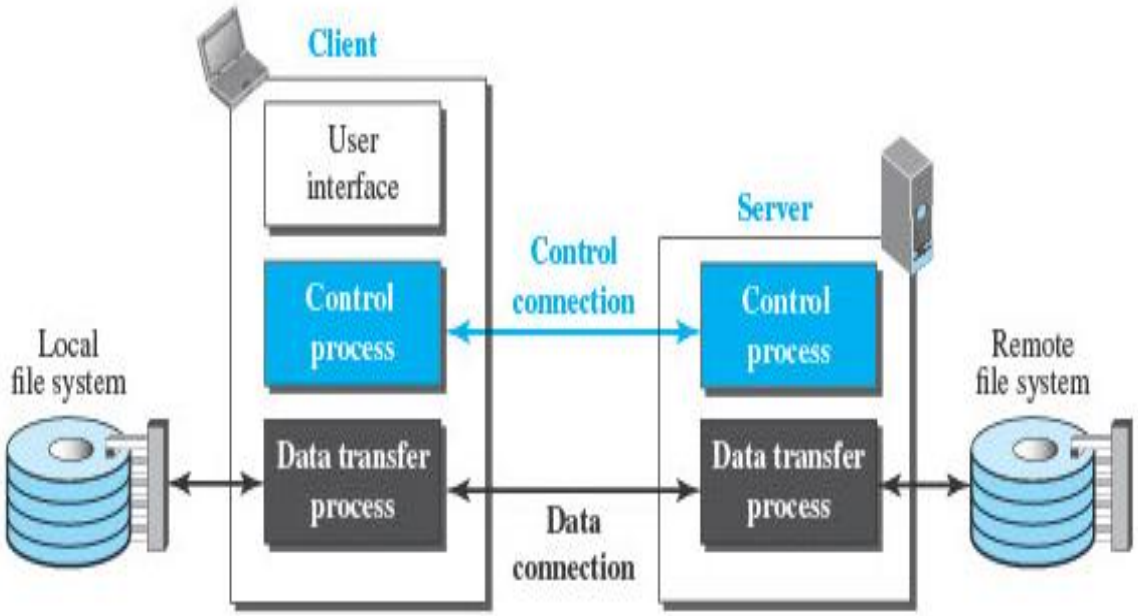
Separation of commands and data transfer makes FTP more efficient.

The control connection uses very simple rules of communication.

We need to transfer only a line of command or a line of response at a time.

The data connection, on the other hand, needs more complex rules due to the variety of data types transferred.

Figure 26.10 *FTP*



Q9)b Solution :

Electronic mail (or e-mail) allows users to exchange messages.

The nature of this application, however, is different from other applications discussed so far.

In an application such as HTTP or FTP, the server program is running all the time, waiting for a request from a client.

When the request arrives, the server provides the service.

There is a request and there is a response.

In the case of electronic mail, the situation is different.

First, e-mail is considered a one-way transaction.

When Alice sends an email to Bob, she may expect a response, but this is not a mandate.

Bob may or may not respond.

If he does respond, it is another one-way transaction.

Second, it is neither feasible nor logical for Bob to run a server program and wait until someone sends an e-mail to him.

Bob may turn off his computer when he is not using it.

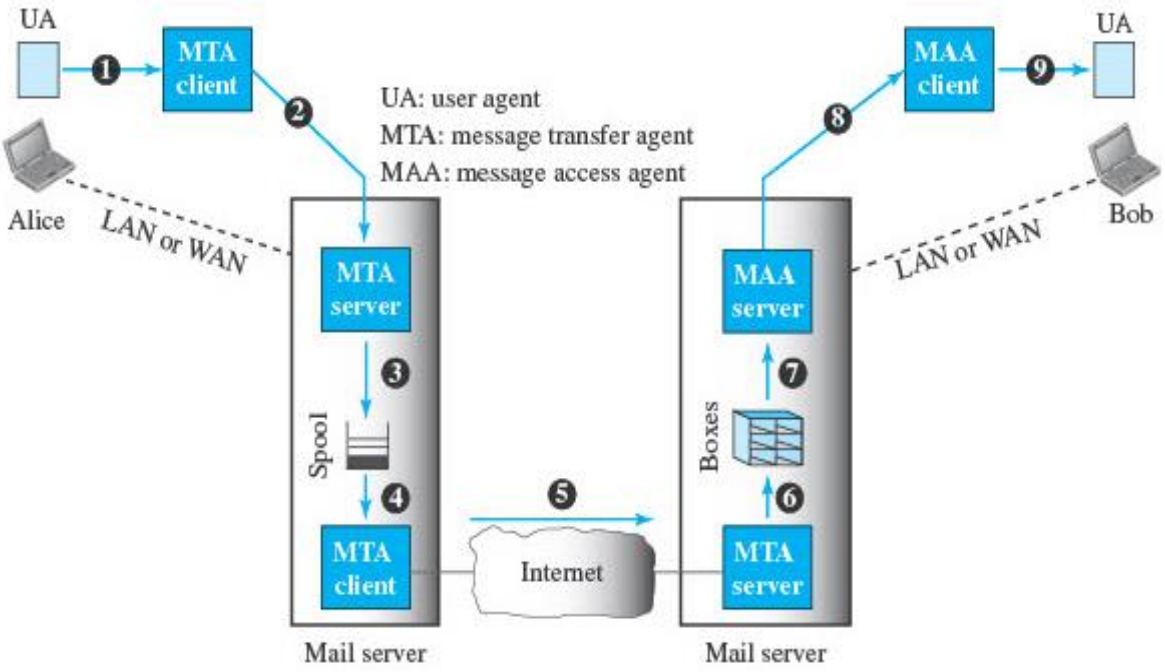
This means that the idea of client/server programming should be implemented in another way: using some intermediate computers (servers).

The users run only client programs when they want and the intermediate servers apply the client/server paradigm, as we discuss in the next section.

To explain the architecture of e-mail, we give a common scenario, as shown in Figure 26.12.

Another possibility is the case in which Alice or Bob is directly connected to the corresponding mail server, in which LAN or WAN connection is not required, but this variation in the scenario does not affect our discussion.

Figure 26.12 *Common scenario*



In the common scenario, the sender and the receiver of the e-mail, Alice and Bob respectively, are connected via a LAN or a WAN to two mail servers.

The administrator has created one mailbox for each user where the received messages are stored.

A mailbox is part of a server hard drive, a special file with permission restrictions.

Only the owner of the mailbox has access to it.

The administrator has also created a queue (spool) to store messages waiting to be sent.

A simple e-mail from Alice to Bob takes nine different steps, as shown in the figure.

Alice and Bob use three different agents: a user agent (UA), a message transfer agent (MTA), and a message access agent (MAA).

When Alice needs to send a message to Bob, she runs a UA program to prepare the message and send it to her mail server.

The mail server at her site uses a queue (spool) to store messages waiting to be sent.

The message, however, needs to be sent through the Internet from Alice's site to Bob's site using an MTA.

Here two message transfer agents are needed: one client and one server.

Like most client-server programs on the Internet, the server needs to run all the time because it does not know when a client will ask for a connection.

The client, on the other hand, can be triggered by the system when there is a message in the queue to be sent.

The user agent at the Bob site allows Bob to read the received message.

Bob later uses an MAA client to retrieve the message from an MAA server running on the second server.

There are two important points we need to emphasize here.

First, Bob cannot bypass the mail server and use the MTA server directly.

To use the MTA server directly, Bob would need to run the MTA server all the time because he does not know when a message will arrive.

This implies that Bob must keep his computer on all the time if he is connected to his system through a LAN.

If he is connected through a WAN, he must keep the connection up all the time.

Neither of these situations is feasible today.

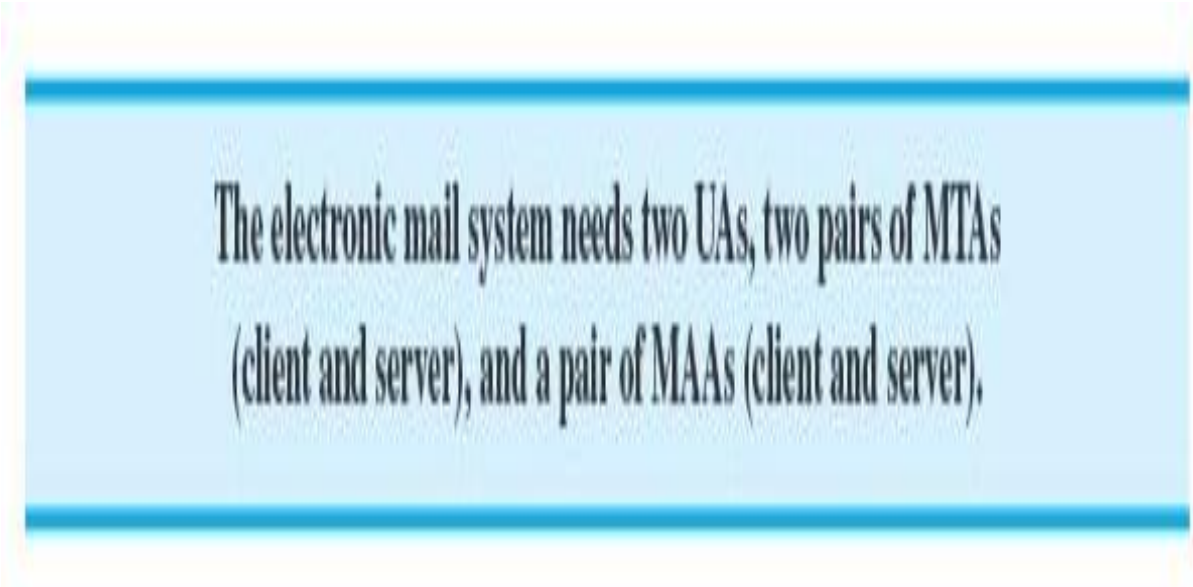
Second, note that Bob needs another pair of client-server programs: message access programs.

This is because an MTA client-server program is a push program: the client pushes the message to the server.

Bob needs a pull program.

The client needs to pull the message from the server.

We discuss more about MAAs shortly.



The first component of an electronic mail system is the user agent (UA).

It provides service to the user to make the process of sending and receiving a message easier.

A user agent is a software package (program) that composes, reads, replies to, and forwards messages.

It also handles local mailboxes on the user computers.

There are two types of user agents: command-driven and GUI-based.

Command-driven user agents belong to the early days of electronic mail.

They are still present as the underlying user agents.

A command-driven user agent normally accepts a one-character command from the keyboard to perform its task.

For example, a user can type the character r, at the command prompt, to reply to the sender of the message, or type the character R to reply to the sender and all recipients.

Some examples of command-driven user agents are mail, pine, and elm.

Modern user agents are GUI-based.

They contain graphical user interface (GUI) components that allow the user to interact with the software by using both the keyboard and the mouse.

They have graphical components such as icons, menu bars, and windows that make the services easy to access.

Some examples of GUI-based user agents are Eudora and Outlook.

Q10)a Solution :

The last client-server application program we discuss has been designed to help other application programs.

To identify an entity, TCP/IP protocols use the IP address, which uniquely identifies the connection of a host to the Internet.

However, people prefer to use names instead of numeric addresses.

Therefore, the Internet needs to have a directory system that can map a name to an address.

This is analogous to the telephone network.

A telephone network is designed to use telephone numbers, not names.

People can either keep a private file to map a name to the corresponding telephone number or can call the telephone directory to do so.

We discuss how this directory system in the Internet can map names to IP addresses.

Since the Internet is so huge today, a central directory system cannot hold all the mapping.

In addition, if the central computer fails, the whole communication network will collapse.

A better solution is to distribute the information among many computers in the world.

In this method, the host that needs mapping can contact the closest computer holding the needed information.

This method is used by the Domain Name System (DNS).

We first discuss the concepts and ideas behind the DNS. We then describe the DNS protocol itself.

Figure 26.28 shows how TCP/IP uses a DNS client and a DNS server to map a name to an address.

A user wants to use a file transfer client to access the corresponding file transfer server running on a remote host.

The user knows only the file transfer server name, such as afilesource.com.

However, the TCP/IP suite needs the IP address of the file transfer server to make the connection.

The following six steps map the host name to an IP address:

1. The user passes the host name to the file transfer client.

2. The file transfer client passes the host name to the DNS client.

3. Each computer, after being booted, knows the address of one DNS server. The DNS client sends a message to a DNS server with a query that gives the file transfer server name using the known IP address of the DNS server.

4. The DNS server responds with the IP address of the desired file transfer server.

5. The DNS server passes the IP address to the file transfer client.

6. The file transfer client now uses the received IP address to access the file transfer server.

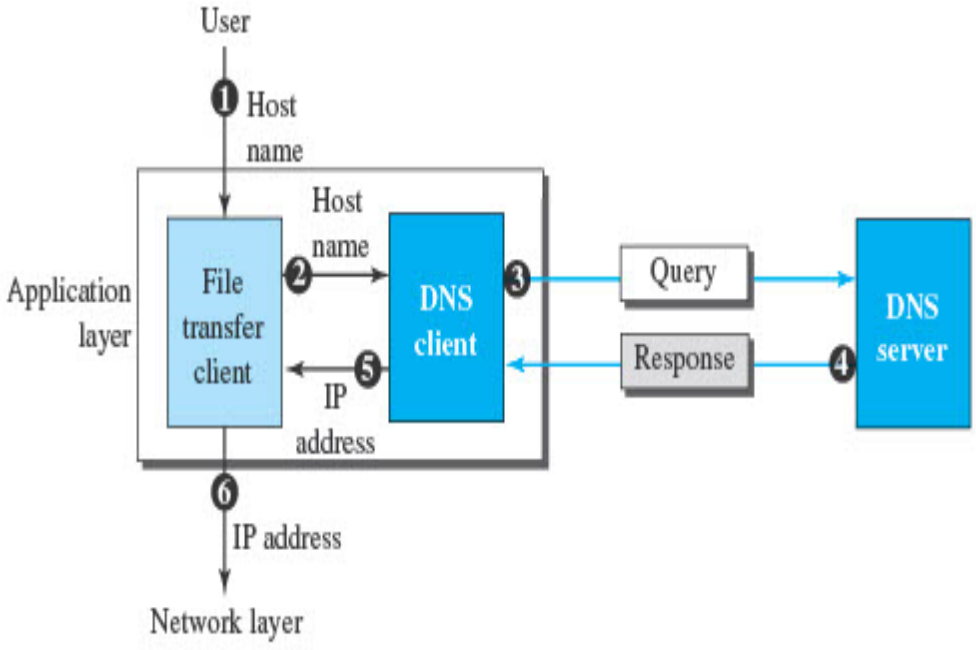
Note that the purpose of accessing the Internet is to make a connection between the file transfer client and server, but before this can happen, another connection needs to be made between the DNS client and DNS server.

In other words, we need at least two connections in this case.

The first is for mapping the name to an IP address; the second is for transferring files.

We will see later that the mapping may need more than one connection.

Figure 26.28 *Purpose of DNS*



To be unambiguous, the names assigned to machines must be carefully selected from a name space with complete control over the binding between the names and IP addresses.

In other words, the names must be unique because the addresses are unique.

A name space that maps each address to a unique name can be organized in two ways: flat or hierarchical.

In a flat name space, a name is assigned to an address.

A name in this space is a sequence of characters without structure.

The names may or may not have a common section; if they do, it has no meaning.

The main disadvantage of a flat name space is that it cannot be used in a large system such as the Internet because it must be centrally controlled to avoid ambiguity and duplication.

In a hierarchical name space, each name is made of several parts.

The first part can define the nature of the organization, the second part can define the name of an organization, the third part can define departments in the organization, and so on.

In this case, the authority to assign and control the name spaces can be decentralized.

A central authority can assign the part of the name that defines the nature of the organization and the name of the organization.

The responsibility for the rest of the name can be given to the organization itself.

The organization can add suffixes (or prefixes) to the name to define its host or resources.

The management of the organization need not worry that the prefix chosen for a host is taken by another organization because, even if part of an address is the same, the whole address is different.

For example, assume two organizations call one of their computers caesar.

The first organization is given a name by the central authority, such as first.com, the second organization is given the name second.com.

When each of these organizations adds the name caesar to the name they have already been given, the end result is two distinguishable names: ceasar.first.com and ceasar.second.com.

The names are unique.

DNS in the Internet

DNS is a protocol that can be used in different platforms.

In the Internet, the domain name space (tree) was originally divided into three different sections: generic domains, country domains, and the inverse domains.

However, due to the rapid growth of the Internet, it became extremely difficult to keep track of the inverse domains, which could be used to find the name of a host when given the IP address.

The inverse domains are now deprecated (see RFC 3425).

We, therefore, concentrate on the first two.

Resolution

Mapping a name to an address is called name-address resolution.

DNS is designed as a client-server application.

A host that needs to map an address to a name or a name to an address calls a DNS client called a resolver.

The resolver accesses the closest DNS server with a mapping request.

If the server has the information, it satisfies the resolver; otherwise, it either refers the resolver to other servers or asks other servers to provide the information.

After the resolver receives the mapping, it interprets the response to see if it is a real resolution or an error, and finally delivers the result to the process that requested it.

A resolution can be either recursive or iterative.

Figure 26.36 shows a simple example of a recursive resolution.

We assume that an application program running on a host named some.anet.com needs to find the IP address of another host named engineering.mcgraw-hill.com to send a message to.

The source host is connected to the Anet ISP; the destination host is connected to the McGraw-Hill network.

The application program on the source host calls the DNS resolver (client) to find the IP address of the destination host.

The resolver, which does not know this address, sends the query to the local DNS server (for example, dns.anet.com) running at the Anet ISP site (event 1).

We assume that this server does not know the IP address of the destination host either.

It sends the query to a root DNS server, whose IP address is supposed to be known to this local DNS server (event 2).

Root servers do not normally keep the mapping between names and IP addresses, but a root server should at least know about one server at each top level domain (in this case, a server responsible for com domain).

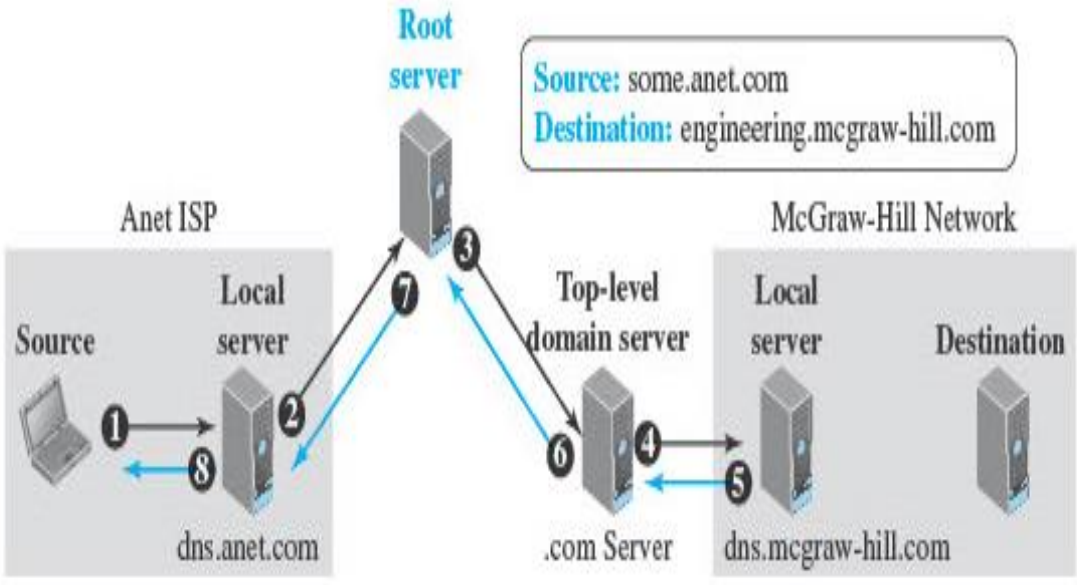
The query is sent to this top-level-domain server (event 3).

We assume that this server does not know the name-address mapping of this specific destination, but it knows the IP address of the local DNS server in the McGraw-Hill company (for example, dns.mcgraw-hill.com).

The query is sent to this server (event 4), which knows the IP address of the destination host.

The IP address is now sent back to the top-level DNS server (event 5), then back to the root server (event 6), then back to the ISP DNS server, which may cache it for the future queries (event 7), and finally back to the source host (event 8).

Figure 26.36 *Recursive resolution*



In iterative resolution, each server that does not know the mapping sends the IP address of the next server back to the one that requested it.

Figure 26.37 shows the flow of information in an iterative resolution in the same scenario as the one depicted in Figure 26.36.

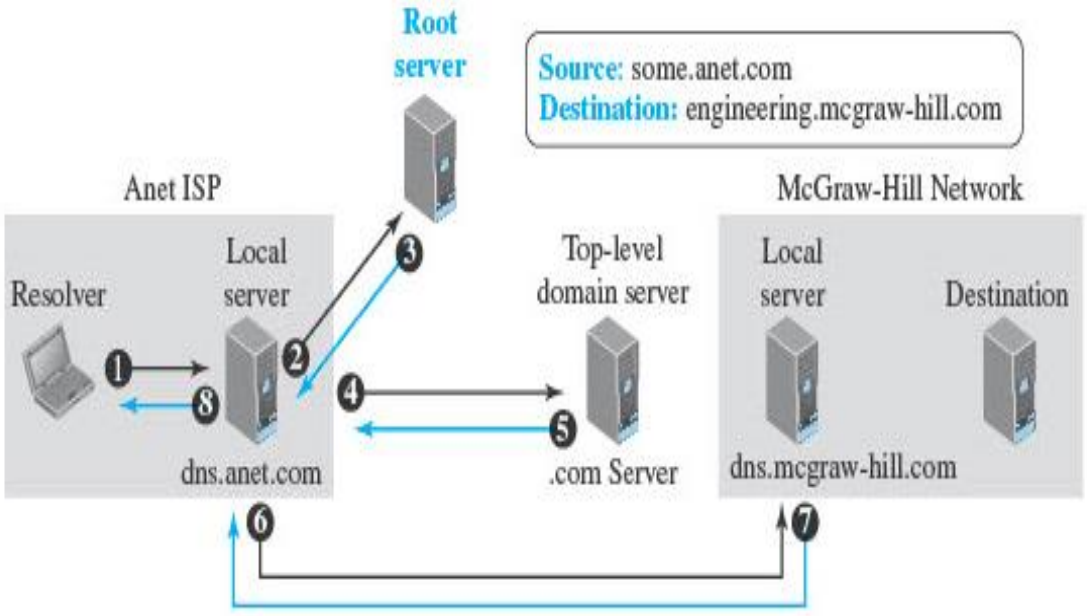
Normally the iterative resolution takes place between two local servers; the original resolver gets the final answer from the local server.

Note that the messages shown by events 2, 4, and 6 contain the same query.

However, the message shown by event 3 contains the IP address of the top-level domain server, the message shown by event 5 contains the IP address of the McGraw-Hill local DNS server, and the message shown by event 7 contains the IP address of the destination.

When the Anet local DNS server receives the IP address of the destination, it sends it to the resolver (event 8).

Figure 26.37 *Iterative resolution*



Q10)b Solution :

[Out of Syllabus].