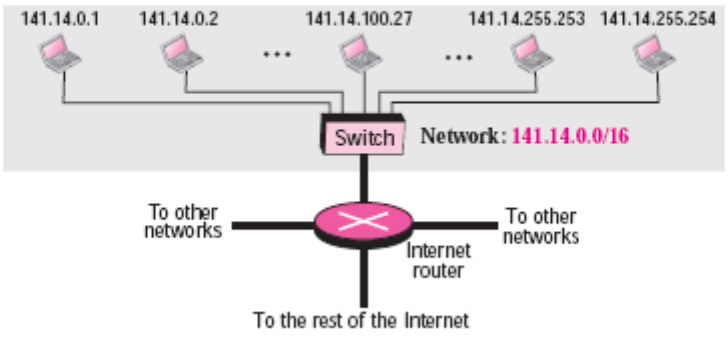
(2½ hours)

Total Marks: 75

N. B.: (1) **All** questions are **compulsory**.
(2) Make **suitable assumptions** wherever necessary and **state the assumptions** made.
(3) Answers to the **same question** must be **written together**.
(4) Numbers to the **right** indicate **marks**.
(5) Draw **neat labeled diagrams** wherever **necessary**.
(6) Use of **Non-programmable** calculators is **allowed**.

1.	**Attempt *any two* of the following:**	**10**
a.	**Why do you need subnetting in classful addressing?** The IP addresses were originally designed with two levels of addressing. To reach a host on the Internet, we must first reach the network and then the host. It soon became clear that we need more than two hierarchical levels, for two reasons. First, an organization that was granted a block in class A or B needed to divide its large network into several subnetworks for better security and management. Second, since the blocks in class A and B were almost depleted and the blocks in class C were smaller than the needs of most organizations, an organization that has been granted a block in class A or B could divide the block into smaller subblocks and share them with other organizations. The idea of splitting a block to smaller blocks is referred to as subnetting. In **subnetting,** a network is divided into several smaller subnetworks (subnets) with each subnetwork having its own subnetwork address. Example 5.18 Figure 5.23 shows a network using class B addresses before subnetting. We have just one network with almost 216 hosts. The whole network is connected, through one single connection, to one of the routers in the Internet. Note that we have shown /16 to show the length of the netid (class B). **Figure 5.23** *Example 5.18*  Subnetting increases the length of the netid and decreases the length of hostid. When we divide a network to s number of subnetworks, each of equal numbers of hosts, we can calculate the subnetid for each subnetwork as $n_{sub} = n + \log_2 s$ in which n is the length of netid, n sub is the length of each subnetid, and s is the number of subnets which must be a power of 2.	
b.	**Why do you need to fragment an IP datagram? Explain the fields related to it.** A datagram can travel through different networks. Each router decapsulates the IP	

Datagram from the frame it receives, processes it, and then encapsulates it in another frame. The format and size of the received frame depend on the protocol used by the physical network through which the frame has just traveled. The format and size of the sent frame depend on the protocol used by the physical network through which the frame is going to travel. For example, if a router connects a LAN to a WAN, it receives a frame in the LAN format and sends a frame in the WAN format.

Maximum Transfer Unit (MTU)

Each data link layer protocol has its own frame format in most protocols. One of the fields defined in the format is the maximum size of the data field. In other words, when a datagram is encapsulated in a frame, the total size of the datagram must be less than this maximum size, which is defined by the restrictions imposed by the hardware and software used in the network.

The value of the MTU differs from one physical network protocol to another. For example, the value for the Ethernet LAN is 1500 bytes, for FDDI LAN is 4352 bytes, and for PPP is 296 bytes.

In order to make the IP protocol independent of the physical network, the designers decided to make the maximum length of the IP datagram equal to 65,535 bytes. This makes transmission more efficient if we use a protocol with an MTU of this size. However, for other physical networks, we must divide the datagram to make it possible to pass through these networks. This is called **fragmentation**.

The source usually does not fragment the IP packet. The transport layer will instead segment the data into a size that can be accommodated by IP and the data link layer in use.

Fields Related to Fragmentation

The fields that are related to fragmentation and reassembly of an IP datagram are the identification, flags, and fragmentation offset fields.

Identification.

This 16-bit field identifies a datagram originating from the source host. The combination of the identification and source IP address must uniquely define a datagram as it leaves the source host. To guarantee uniqueness, the IP protocol uses a counter to label the datagrams. The counter is initialized to a positive number. When the IP protocol sends a datagram, it copies the current value of the counter to the identification field and increments the counter by one. As long as the counter is kept in the main memory, uniqueness is guaranteed. When a datagram is fragmented, the value in the identification field is copied into all fragments. In other words, all fragments have the same identification number, which is also the same as the original datagram. The identification number helps the destination in reassembling the datagram. It knows that all fragments having the same identification value should be assembled into one datagram.

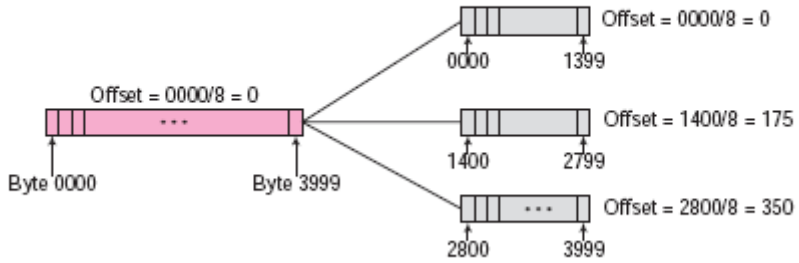
Flags.

This is a three-bit field. The first bit is reserved (not used). The second bit is called the do not fragment bit. If its value is 1, the machine must not fragment the datagram. If it cannot pass the datagram through any available physical network, it discards the datagram and sends an ICMP error message to the source host (see Chapter 9). If its value is 0, the datagram can be fragmented if necessary. The third bit is called the more fragment bit. If its value is 1, it means the datagram is not the last fragment; there are more fragments after this one. If its value is 0, it means this is the last or only fragment (see Figure 7.7).

Figure 7.7 *Flags field*

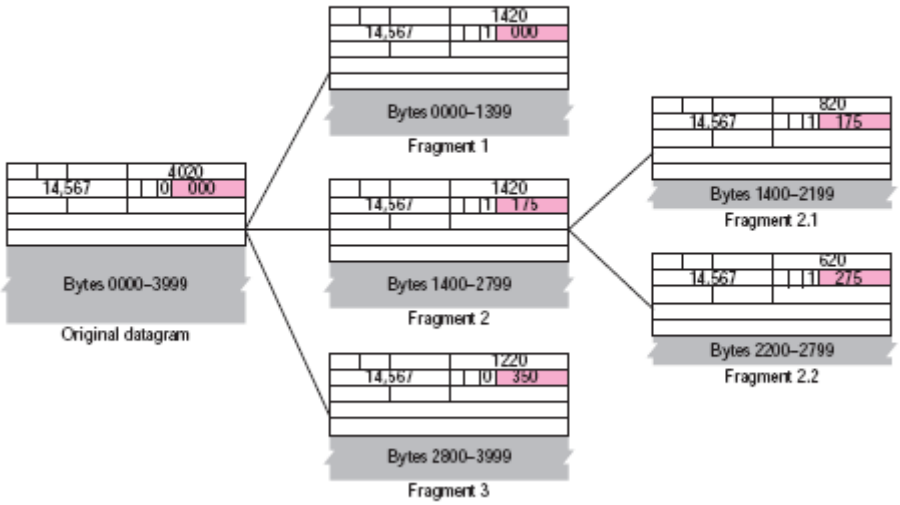
Fragmentation offset.
This 13-bit field shows the relative position of this fragment with respect to the whole datagram. It is the offset of the data in the original datagram measured in units of 8 bytes. Figure 7.8 shows a datagram with a data size of 4000 bytes fragmented into three fragments. The bytes in the original datagram are numbered 0 to 3999. The first fragment carries bytes 0 to 1399. The offset for this datagram is 0/8 = 0. The second fragment carries bytes 1400 to 2799; the offset value for this fragment is 1400/8 = 175. Finally, the third fragment carries bytes 2800 to 3999. The offset value for this fragment is 2800/8 = 350.

Figure 7.8 *Fragmentation example*



Remember that the value of the offset is measured in units of 8 bytes. This is done because the length of the offset field is only 13 bits long and cannot represent a sequence of bytes greater than 8191. This forces hosts or routers that fragment datagrams to choose the size of each fragment so that the first byte number is divisible by 8.
Figure 7.9 shows an expanded view of the fragments in the previous figure. Notice the value of the identification field is the same in all fragments. Notice the value of the flags field with the more bit set for all fragments except the last. Also, the value of the offset field for each fragment is shown.

Figure 7.9 *Detailed fragmentation example*



The figure also shows what happens if a fragment itself is fragmented. In this case the value of the offset field is always relative to the original datagram. For example, in the figure, the second fragment is itself fragmented later to two fragments of 800 bytes and 600 bytes, but the offset shows the relative position of the fragments to the original data. It is obvious that even if each fragment follows a different path and arrives out of order, the final destination host can reassemble the original datagram from the fragments received (if none of them is lost) using the following strategy:
a. The first fragment has an offset field value of zero.
b. Divide the length of the first fragment by 8. The second fragment has an offset value equal to that result.
c. Divide the total length of the first and second fragment by 8. The third fragment has an offset value equal to that result.
d. Continue the process. The last fragment has a more bit value of 0.

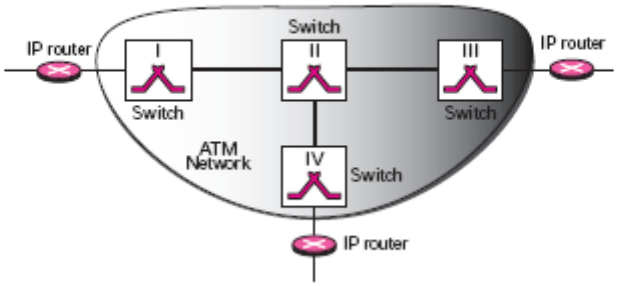
c. **Explain the concept of IP over ATM.**

The IP packet is encapsulated in cells (not just one). An ATM network has its own definition for the physical address of a device. Binding between an IP address and a physical address is attained through a protocol called ATMARP.

ATM WANs

We discussed ATM WANs in Chapter 3. ATM, a cell-switched network, can be a highway for an IP datagram. Figure 7.26 shows how an ATM network can be used in the Internet.

Figure 7.26 *An ATM WAN in the Internet*



AAL Layer

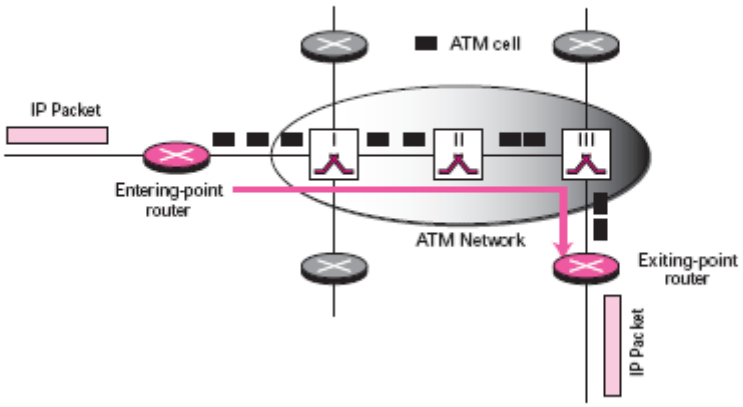
The only AAL used by the Internet is AAL5. It is sometimes called the simple and efficient adaptation layer (SEAL). AAL5 assumes that all cells created from one IP datagram belong to a single message. AAL5 therefore provides no addressing, sequencing, or other header information. Instead, only padding and a four-field trailer are added to the IP packet. AAL5 accepts an IP packet of no more than 65,536 bytes and adds an 8-byte trailer as well as any padding required to ensure that the position of the trailer falls where the receiving equipment expects it (at the last 8 bytes of the last cell). Once the padding and trailer are in place, AAL5 passes the message in 48-byte segments to the ATM layer.

The AAL layer used by the IP protocol is AAL5.

Why Use AAL5?

A question that frequently comes up is why do we use AAL5. Why can't we just encapsulate an IP packet in a cell? The answer is that it is more efficient to use AAL5. If an IP datagram is to be encapsulated in a cell, the data at the IP level must be 53 − 5 − 20 = 27 bytes because a minimum of 20 bytes is needed for the IP header and 5 bytes is needed for the ATM header. The efficiency is 27/53, or almost 51 percent. By letting an IP datagram span over several cells, we are dividing the IP overhead (20 bytes) among those cells and increasing efficiency. Routing the Cells The ATM network creates a route between two routers. We call these routers entering point and exiting-point routers. The cells start from the entering-point router and end at the exiting-point router as shown in Figure 7.27.

Figure 7.27 *Entering-point and exiting-point routers*



Addresses

Routing the cells from one specific entering-point router to one specific exiting-point router requires three types of addressing: IP addresses, physical addresses, and virtual circuit identifiers.

IP Addresses Each router connected to the ATM network has an IP address. Later we will see that the addresses may or may not have the same prefix. The IP address defines the router at the IP layer. It does not have anything to do with the ATM network.

Physical Addresses Each router (or any other device) connected to the ATM network has also a physical address. The physical address is associated with the ATM network and does not have anything to do with the Internet. The ATM Forum defines 20-byte addresses for ATM networks. Each address must be unique in a network and is defined by the network administrator. The physical addresses in an ATM network play the same role as the MAC addresses in a LAN. The physical addresses are used during connection establishment.

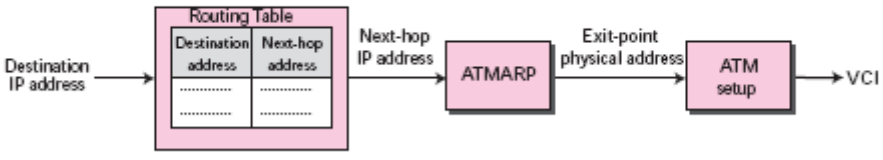
Virtual Circuit Identifiers The switches inside the ATM network route the cells based on the virtual circuit identifiers (VPIs and VCIs), as we discussed in Chapter 3. The virtual circuit identifiers are used during data transfer.

Address Binding

An ATM network needs virtual circuit identifiers to route the cells. The IP datagram contains only source and destination IP addresses. Virtual circuit identifiers must be determined from the destination IP address. Figure 7.28 shows how this is done. These are the steps:

1. The entering-point router receives an IP datagram. It uses the destination address and its routing table to find the IP address of the next router, the exiting-point router. This is exactly the same step followed when a datagram passes through a LAN.

Figure 7.28 *Address binding in IP over ATM*



2. The entering-point router uses the services of a protocol called ATMARP to find the physical address of the exiting-point router. ATMARP is similar to ARP
3. The virtual circuit identifiers are bound to the physical addresses.

d.	**Explain types of extension headers in IPv6.**	

The length of the base header is fixed at 40 bytes. However, to give more functionality to the IP datagram, the base header can be followed by up to six
Extension headers. Many of these headers are options in IPv4. Figure 27.3 shows the extension header format.
Six types of extension headers have been defined. These are hop-by-hop option, source routing, fragmentation, authentication, encrypted security payload, and destination option (see Figure 27.4).

Hop-by-Hop Option

The hop-by-hop option is used when the source needs to pass information to all routers visited by the datagram. For example, perhaps routers must be informed about certain management, debugging, or control functions. Or, if the length of the datagram is more than the usual 65,535 bytes, routers must have this information. Figure 27.5 shows the format of the hop-by-hop option header. The first field defines the next header in the chain of headers. The header length defines the number of bytes in the header (including the next header field). The rest of the header contains different options.

Figure 27.3 *Extension header format*

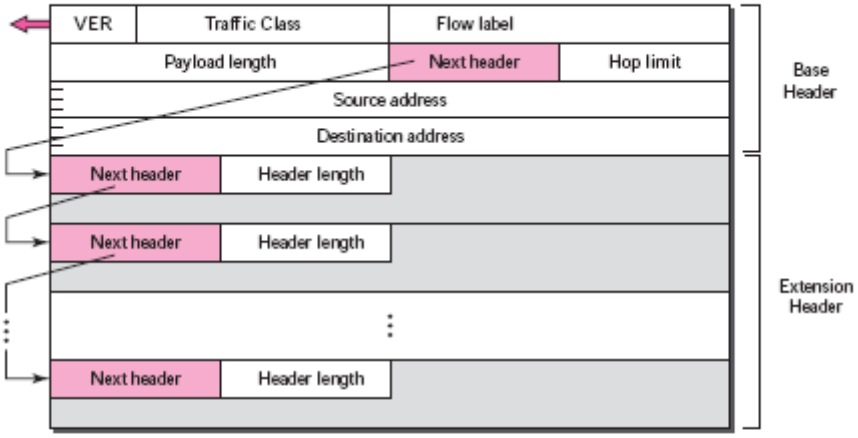


Figure 27.4 *Extension header types*

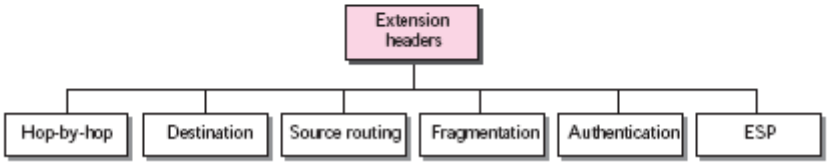
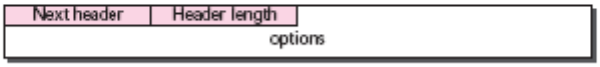


Figure 27.5 *Hop-by-hop option header format*

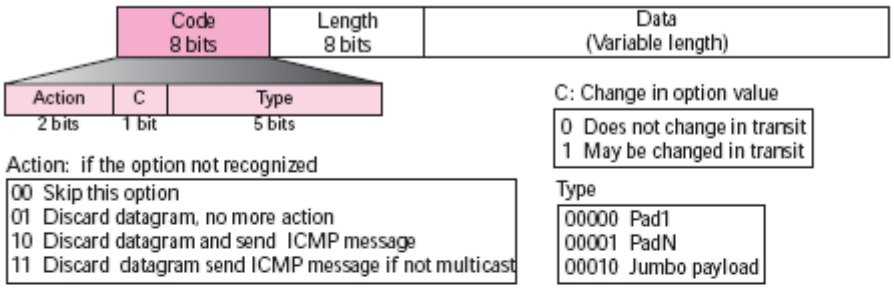


So far, only three hop-by-hop options have been defined:
Pad1, PadN, and jumbo payload. Figure 27.6 shows the general format of the option.

Pad1.

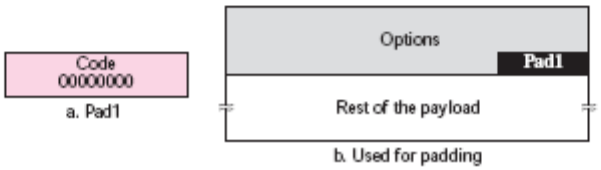
This option is 1 byte long and is designed for alignment purposes. Some options need to start at a specific bit of the 32-bit word (see the jumbo payload description to come). If an option falls short of this requirement by exactly one

Figure 27.6 *The format of options in a hop-by-hop option header*



byte, Pad1 is added to make up the difference. Pad1 contains neither the option length field nor the option data field. It consists solely of the option code field with all bits set to 0 (action is 00, the change bit is 0, and type is 00000). Pad1 can be inserted anywhere in the hop-by-hop option header (see Figure 27.7).

Figure 27.7 *Pad1*

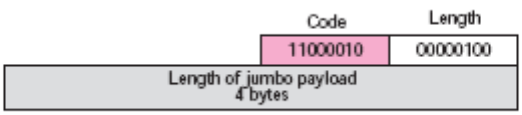


❑ **PadN.** PadN is similar in concept to Pad1. The difference is that PadN is used when 2 or more bytes are needed for alignment. This option consists of 1 byte of option code, 1 byte of the option length, and a variable number of zero padding bytes. The value of the option code is 1 (action is 00, the change bit is 0, and type is 00001). The option length contains the number of padding bytes. See Figure 27.8.

Figure 27.8 *PadN*

❑ **Jumbo payload.** Recall that the length of the payload in the IP datagram can be a maximum of 65,535 bytes. However, if for any reason a longer payload is required, we can use the jumbo payload option to define this longer length. The jumbo payload option must always start at a multiple of 4 bytes plus 2 from the beginning of the extension headers. The jumbo payload option starts at the $(4n + 2)$ byte, where n is a small integer. See Figure 27.9.

Figure 27.9 *Jumbo payload*



. **Destination Option**

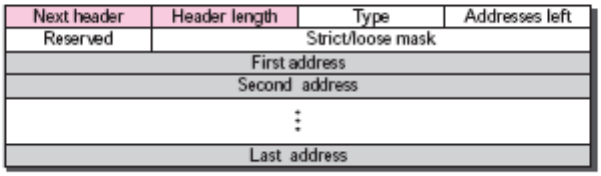
The destination option is used when the source needs to pass information to the destination only. Intermediate routers are not permitted access to this information. The format of the destination option is the same as the hop-by-hop option (refer back to Figure 27.5). So far, only the Pad1 and PadN options have been defined.

Source Routing

The source routing extension header combines the concepts of the strict source route

and the loose source route options of IPv4. The source routing header contains a minimum of seven fields (see Figure 27.10). The first two fields, next header and header length, are identical to that of the hop-by-hop extension header. The type field defines loose or strict routing. The addresses left field indicates the number of hops still needed to reach the destination. The strict/loose mask field determines the rigidity of routing. If set to strict, routing must follow exactly as indicated by the source. If, instead, the mask is loose, other routers may be visited in addition to those in the header.

Figure 27.10 *Source routing*

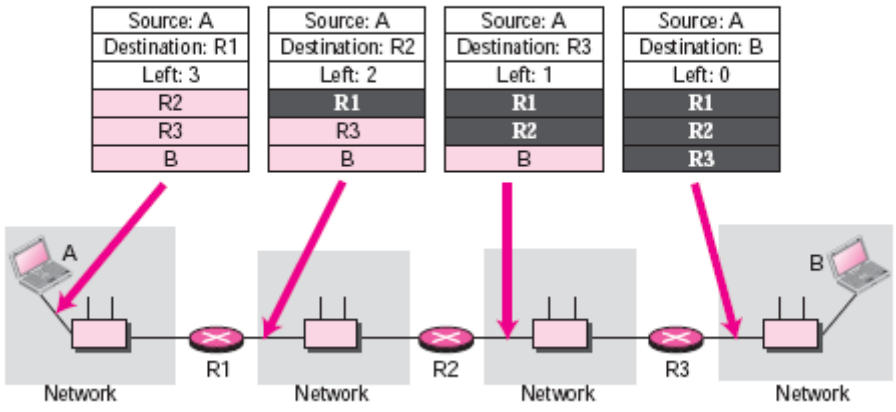


The destination address in source routing does not conform to our previous definition (the final destination of the datagram). Instead, it changes from router to router. For example, in Figure 27.11, Host A wants to send a datagram to Host B using a specific route: A to R1 to R2 to R3 to B. Notice the destination address in the base headers. It is not constant as you might expect. Instead, it changes at each router. The addresses in the extension headers also change from router to router.

Fragmentation

The concept of fragmentation is the same as that in IPv4. However, the place where fragmentation occurs differs. In IPv4, the source or a router is required to fragment if

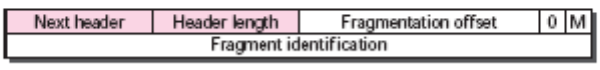
Figure 27.11 *Source routing example*



the size of the datagram is larger than the MTU of the network over which the datagram travels. In IPv6, only the original source can fragment. A source must use a Path MTU Discovery technique to find the smallest MTU supported by any network on the path. The source then fragments using this knowledge.

If the source does not use a Path MTU Discovery technique, it fragments the datagram to a size of 1,280 bytes or smaller. This is the minimum size of MTU required for each network connected to the Internet. Figure 27.12 shows the format of the fragmentation extension header.

Figure 27.12 *Fragmentation*



Authentication

The authentication extension header has a dual purpose: it validates the message sender and ensures the integrity of data. The former is needed so the receiver can be sure that a message is from the genuine sender and not from an imposter. The latter is needed to

check that the data is not altered in transition by some hacker.

The format of the authentication extension header is shown in Figure 27.13. The security parameter index field defines the algorithm used for authentication. The authentication data field contains the actual data generated by the algorithm. We will discuss authentication in Chapter 29.

Many different algorithms can be used for authentication. Figure 27.14 outlines the method for calculating the authentication data field. The sender passes a 128-bit security key, the entire IP datagram, and the 128-bit security key again to the algorithm. Those fields in the datagram with values that change during transmission (for example, hop count)

Figure 27.13 *Authentication*

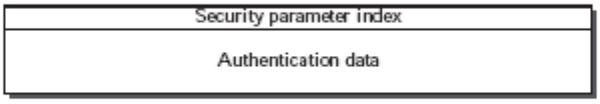


Figure 27.14 *Calculation of authentication data*

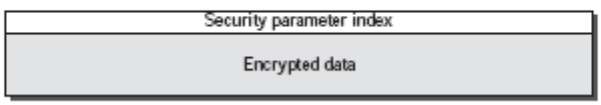
are set to zero. The datagram passed to the algorithm includes the authentication header extension, with the authentication data field set to zero. The algorithm creates authentication data which is inserted into the extension header prior to datagram transmission.

The receiver functions in a similar manner. It takes the secret key and the received datagram (again, with changeable fields set to zero) and passes them to the authentication algorithm. If the result matches that in the authentication data field, the IP datagram is authentic; otherwise, the datagram is discarded.

Encrypted Security Payload

The encrypted security payload (ESP) is an extension that provides confidentiality and guards against eavesdropping. Figure 27.15 shows the format. The security parameter index field is a 32-bit word that defines the type of encryption/decryption used. The other field contains the encrypted data along with any extra parameters needed by the algorithm. Encryption can be implemented in two ways: transport mode or tunnel mode

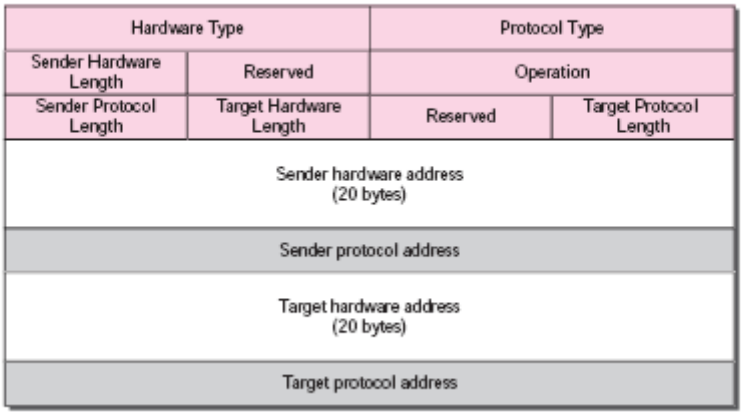
Figure 27.15 *Encrypted security payload*



.

2.	**Attempt *any two* of the following:**	**10**
a.	**Explain the packet format of ATMARP**. **Packet Format** The format of an ATMARP packet, which is similar to the ARP packet, is shown in below diagram.	

Figure 8.8 *ATMARP packet*



The fields are as follows:

❑**Hardware type (HTYPE)**. The 16-bit HTYPE field defines the type of the physical Network. Its value is 0013_{16} for an ATM network.

❑**Protocol type (PTYPE)**. The 16-bit PTYPE field defines the type of the protocol. For IPv4 protocol the value is 0800_{16}.

❑**Sender hardware length (SHLEN).** The 8-bit SHLEN field defines the length of the sender's physical address in bytes. For an ATM network the value is 20. Note that if the binding is done across an ATM network and two levels of hardware addressing are necessary, the neighboring 8-bit **reserved field** is used to define the length of the second address.

❑**Operation (OPER).** The 16-bit OPER field defines the type of the packet. Five packet types are defined as shown in Table 8.1.

Table 8.1 *OPER field*

Message	*OPER value*
Request	1
Reply	2
Inverse Request	8
Inverse Reply	9
NACK	10

❑**Sender protocol length (SPLEN).** The 8-bit SPLEN field defines the length of the address in bytes. For IPv4 the value is 4 bytes.

❑**Target hardware length (TLEN).** The 8-bit TLEN field defines the length of the receiver's physical address in bytes. For an ATM network the value is 20. Note that if the binding is done across an ATM network and two levels of hardware addressing are necessary, the neighboring 8-bit reserved field is used to define the length of the second address.

❑**Target protocol length (TPLEN).** The 8-bit TPLEN field defines the length of the address in bytes. For IPv4 the value is 4 bytes.

❑**Sender hardware address (SHA).** The variable-length SHA field defines the physical address of the sender. For ATM networks defined by the ATM Forum, the length is 20 bytes.

❑**Sender protocol address (SPA).** The variable-length SPA field defines the address of the sender. For IPv4 the length is 4 bytes.

❑**Target hardware address (THA).** The variable-length THA field defines the physical address of the receiver. For ATM networks defined by the ATM Forum, the length is 20 bytes. This field is left empty for request messages and filled in for reply and NACK messages.

❑**Target protocol address (TPA).** The variable-length TPA field defines the address of the receiver. For IPv4 the length is 4 bytes.

b. **List the error reporting messages in ICMP? Explain any 1 of them with its format.**

Figure 9.4 *Error-reporting messages*

Parameter Problem

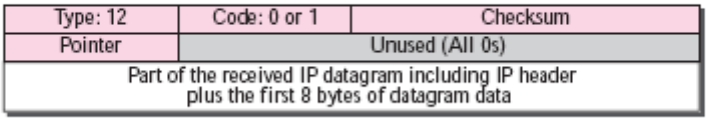
Any ambiguity in the header part of a datagram can create serious problems as the datagram travels through the Internet. If a router or the destination host discovers an ambiguous or missing value in any field of the datagram, it discards the datagram and sends a parameter-problem message back to the source.

Figure 9.9 shows the format of the **parameter-problem message.** The code field in this case specifies the reason for discarding the datagram:

❑**Code 0.** There is an error or ambiguity in one of the header fields. In this case, the value in the pointer field points to the byte with the problem. For example, if the value is zero, then the first byte is not a valid field.

❑**Code 1.** The required part of an option is missing. In this case, the pointer is not used.

Figure 9.9 *Parameter-problem message format*



Note: A student can explain any one error reporting message. I have shown one of them in sample answer.

c. **<u>What is the inefficiency in mobile IP? Explain with the solution</u>**.

INEFFICIENCY IN MOBILE IP

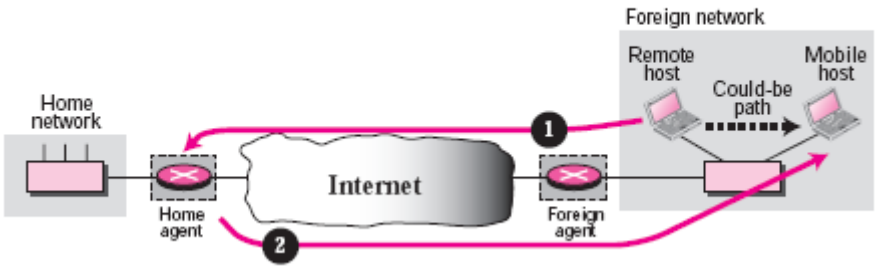
Communication involving mobile IP can be inefficient. The inefficiency can be severe or moderate. The severe case is called *double crossing* or *2X* . The moderate case is called *triangle routing* or *dog-leg routing*

.

Double Crossing

Double crossing occurs when a remote host communicates with a mobile host that has moved to the same network (or site) as the remote host (see Figure 10.8).

Figure 10.8 *Double crossing*



When the mobile host sends a packet to the remote host, there is no inefficiency; the communication is local. However, when the remote host sends a packet to the mobile host, the packet crosses the Internet twice. Since a computer usually communicates with other local computers (principle of locality), the inefficiency from double crossing is significant.

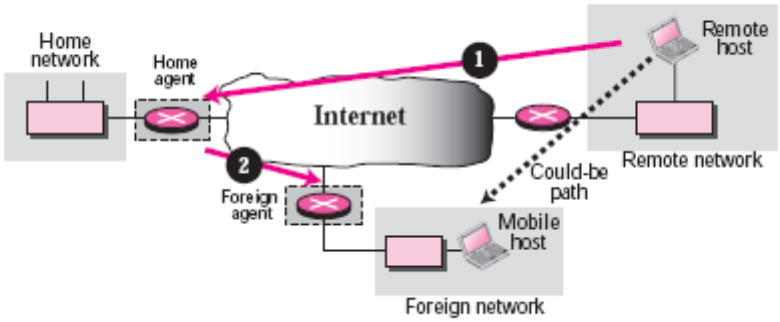
Triangle Routing

Triangle routing, the less severe case, occurs when the remote host communicates with a mobile host that is not attached to the same network (or site) as the mobile host. When the mobile host sends a packet to the remote host, there is no inefficiency. However, when the remote host sends a packet to the mobile host, the packet goes from the remote host to the home agent and then to the mobile host. The packet travels the two sides of a triangle, instead of just one side (see Figure 10.9).

Solution

One solution to inefficiency is for the remote host to bind the care-of address to the home address of a mobile host. For example, when a home agent receives the first packet for a mobile host, it forwards the packet to the foreign agent; it could also send an **update binding packet** to the remote host so that future packets to this host could be sent to the care-of address. The remote host can keep this information in a cache. The problem with this strategy is that the cache entry becomes outdated once the mobile host moves. In this case the home agent needs to send a **warning packet** to the remote host to inform it of the change.

Figure 10.9 *Triangle routing*

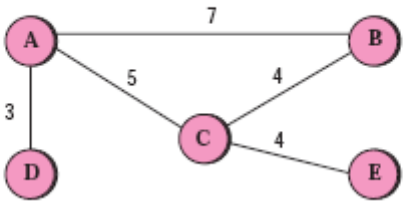


d. **How Bellman-Ford algorithm helps to find least cost between any two nodes?**

Bellman-Ford Algorithm

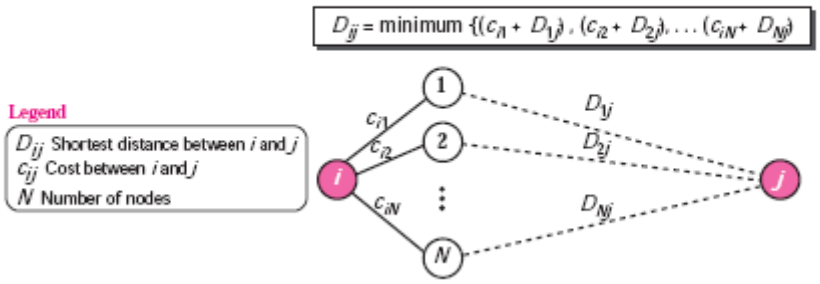
The algorithm can be used in many applications in graph theory. If we know the cost between each pair of nodes, we can use the algorithm to find the least cost (shortest path) between any two nodes. Figure 11.3 shows a map with nodes and lines. The cost of each line is given over the line; the algorithm can find the least cost between any two nodes. For example, if the nodes represent cities and the lines represent roads connecting them, the graph can find the shortest distance between any two cities.

Figure 11.3 *A graph for the Bellman-Ford algorithm*



The algorithm is based on the fact that if all neighbors of node I know the shortest distance to node j , then the shortest distance between node I and j can be found by adding the distance between node i and each neighbor to the neighbor's shortest distance to node j and then select the minimum, as shown in Figure 11.4.

Figure 11.4 *The fact behind Bellman-Ford algorithm*



Although the principle of the Bellman-Ford algorithm is very simple and intuitive, the algorithm itself looks circular. How have the neighbors calculated the shortest path to the destination? To solve the problem, we use iteration. We create a shortest distance table (vector) for each node using the following steps:

1. The shortest distance and the cost between a node and itself is initialized to 0.

2. The shortest distance between a node and any other node is set to infinity. The cost between a node and any other node should be given (can be infinity if the nodes are not connected).

3. The algorithm repeat as shown in Figure 11.4 until there is no more change in the shortest distance vector. Table 11.1 shows the algorithm in pseudocode.

Table 11.1 *Bellman-Ford Algorithm*

```
1  Bellman_Ford ( )
2  {
3     // Initialization
4     for (i = 1 to N; for j = 1 to N)
5     {
6        if(i == j)   D_ij = 0   c_ij = 0
7        else         D_ij = ∞   c_ij = cost between i and j
8     }
9     // Updating
10    repeat
11    {
12       for (i = 1 to N; for j = 1 to N)
13       {
14          D_ij ← minimum [(c_i1 + D_1j)  ...  (c_iN + D_Nj)]
15       } // end for
16    } until (there was no change in previous iteration)
17 } // end Bellman-Ford
```

3.	**Attempt *any two* of the following:**	**10**
a.	**Explain in detail UDP Package**.	

To show how UDP handles the sending and receiving of UDP packets, we present a simple version of the UDP package. We can say that the UDP package involves five components: a control-block table, input queues, a control-block module, an input module, and an output module. Figure 14.8 shows these five components and their interactions.

Control-Block Table

In our package, UDP has a control-block table to keep track of the open ports. Each entry in this table has a minimum of four fields: the state, which can be FREE or IN-USE, the

process ID, the port number, and the corresponding queue number.

Input Queues

Our UDP package uses a set of input queues, one for each process. In this design, we do not use output queues.

Control-Block Module

The control-block module (Table 14.2) is responsible for the management of the control-block table. When a process starts, it asks for a port number from the operating system. The operating system assigns well-known port numbers to servers and ephemeral port numbers to clients. The process passes the process ID and the port number to the control-block module to create an entry in the table for the process. The module does not create the queues. The field for queue number has a value of zero. Note that we have not included a strategy to deal with a table that is full.

Figure 14.8 *UDP design*

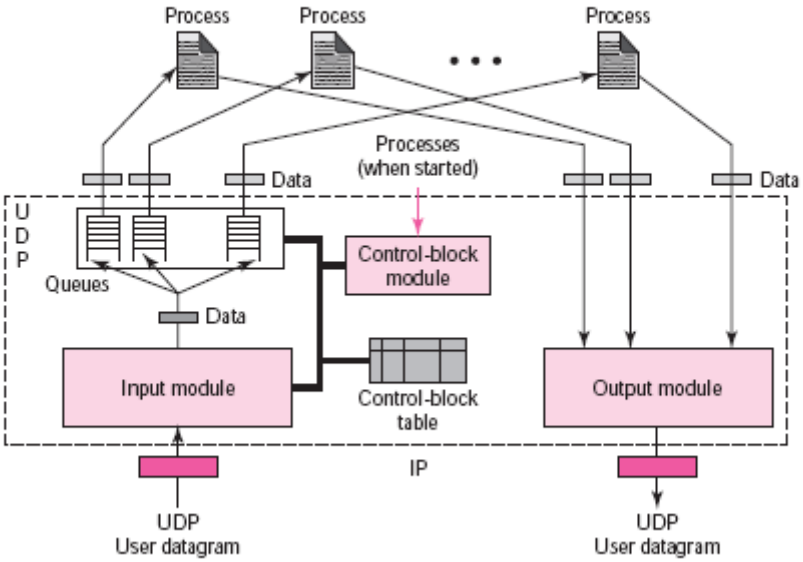


Table 14.2 *Control Block Module*

```
1  UDP_Control_Block_Module (process ID, port number)
2  {
3       Search the table for a FREE entry.
4       if (not found)
5            Delete one entry using a predefined strategy.
6       Create a new entry with the state IN-USE
7       Enter the process ID and the port number.
8       Return.
9  } // End module
```

Input Module

The input module (Table 14.3) receives a user datagram from the IP. It searches the control-block table to find an entry having the same port number as this user datagram. If the entry is found, the module uses the information in the entry to enqueue the data. If the entry is not found, it generates an ICMP message.

Table 14.3 *Input Module*

```
1  UDP_INPUT_Module (user_datagram)
2  {
3       Look for the entry in the control_block table
4       if (found)
5       {
6            Check to see if a queue is allocated
7            If (queue is not allocated)
8                 allocate a queue
9            else
10                enqueue the data
11      } //end if
12      else
13      {
14           Ask ICMP to send an "unreachable port" message
15           Discard the user datagram
16      } //end else
17
18      Return.
19 } // end module
```

Output Module

The output module (Table 14.4) is responsible for creating and sending user datagrams.

Table 14.4 *Output Module*

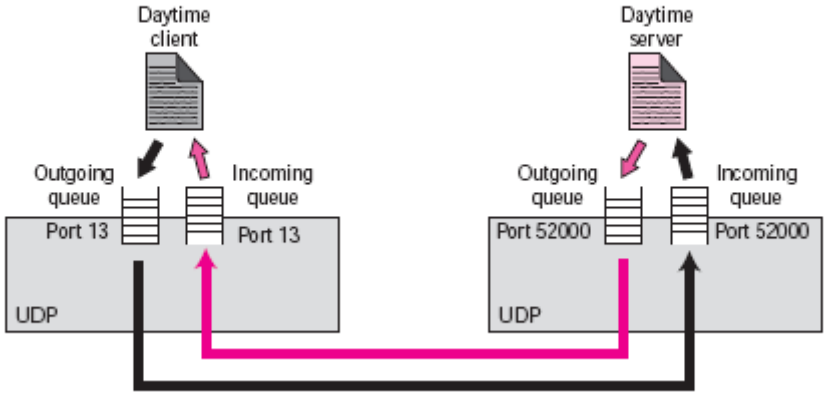
```
1  UDP_OUTPUT_MODULE (Data)
2  {
3       Create a user datagram
4       Send the user datagram
5       Return.
6  }
```

b. **What is the need of queues in UDP?**

Queuing

We have talked about ports without discussing the actual implementation of them. In UDP, queues are associated with ports (see Figure 14.6). 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.

Figure 14.6 *Queues in UDP*



	Note that even if a process wants to communicate with multiple processes, it obtains only one port number and eventually one outgoing and one incoming **queue.** The queues opened by the client are, in most cases, identified by ephemeral port numbers. The queues function as long as the process is running. When the process terminates, the queues are destroyed. The client process can send messages to the outgoing queue by using the source port number specified in the request. UDP removes the messages one by one, and, after adding the UDP header, delivers them to IP. An outgoing queue can overflow. If this happens, the operating system can ask the client process to wait before sending any more messages. When a message arrives for a client, UDP checks to see if an incoming queue has been created for the port number specified in the destination port number field of the user datagram. If there is such a queue, UDP sends the received user datagram to the end of the queue. If there is no such queue, UDP discards the user datagram and asks the ICMP protocol to send a *port unreachable* message to the server. All of the incoming messages for one particular client program, whether coming from the same or a different server, are sent to the same queue. An incoming queue can overflow. If this happens, UDP drops the user datagram and asks for a port unreachable message to be sent to the server. At the server site, the mechanism of creating queues is different. In its simplest form, a server asks for incoming and outgoing queues using its well-known port when it starts running. The queues remain open as long as the server is running. When a message arrives for a server, UDP checks to see if an incoming queue has been created for the port number specified in the destination port number field of the user datagram. If there is such a queue, UDP sends the received user datagram to the end of the queue. If there is no such queue, UDP discards the user datagram and asks the ICMP protocol to send a port unreachable message to the client. All of the incoming messages for one particular server, whether coming from the same or a different client, are sent to the same queue. An incoming queue can overflow. If this happens, UDP drops the user datagram and asks for a port unreachable message to be sent to the client. When a server wants to respond to a client, it sends messages to the outgoing queue using the source port number specified in the request. UDP removes the messages one by one, and, after adding the UDP header, delivers them to IP. An outgoing queue can overflow. If this happens, the operating system asks the server to wait before sending any more messages.	
c.	**<u>Explain Numbering system used in TCP.</u>** **Numbering System** Although the TCP software keeps track of the segments being transmitted or received, there is no field for a segment number value in the segment header. Instead, there are two fields called the *sequence number* and the *acknowledgment number* . These two fields refer to a byte number and not a segment number. ***Byte Number*** TCP numbers all data bytes (octets) that are transmitted in a connection. Numbering is independent in each direction. When TCP receives bytes of data from a process, TCP stores them in the sending buffer and numbers them. The numbering does not necessarily start from 0. Instead, TCP chooses an arbitrary number between 0 and 2^{32}-1 for the number of the first byte. For example, if the number happens to be 1,057 and the total data to be sent is 6,000 bytes, the bytes are numbered from 1,057 to 7,056. We will see that byte numbering is used for flow and error control.	

The bytes of data being transferred in each connection are numbered by TCP. The numbering starts with an arbitrarily generated number.

Sequence Number

After the bytes have been numbered, TCP assigns a sequence number to each segment that is being sent. The sequence number for each segment is the number of the first byte of data carried in that segment.

Example 15.1

Suppose a TCP connection is transferring a file of 5,000 bytes. The first byte is numbered 10,001. What are the sequence numbers for each segment if data are sent in five segments, each carrying 1,000 bytes?

Solution

The following shows the sequence number for each segment:

Segment 1	→	Sequence Number:	10,001	Range:	10,001	to	11,000
Segment 2	→	Sequence Number:	11,001	Range:	11,001	to	12,000
Segment 3	→	Sequence Number:	12,001	Range:	12,001	to	13,000
Segment 4	→	Sequence Number:	13,001	Range:	13,001	to	14,000
Segment 5	→	Sequence Number:	14,001	Range:	14,001	to	15,000

The value in the sequence number field of a segment defines the number assigned to the first data byte contained in that segment.

When a segment carries a combination of data and control information (piggybacking), it uses a sequence number. If a segment does not carry user data, it does not logically define a sequence number. The field is there, but the value is not valid. However, some segments, when carrying only control information, need a sequence number to allow an acknowledgment from the receiver. These segments are used for connection establishment, termination, or abortion. Each of these segments consumes one sequence number as though it carries one byte, but there are no actual data. We will elaborate on this issue when we discuss connections.

Acknowledgment Number

As we discussed previously, communication in TCP is full duplex; when a connection is established, both parties can send and receive data at the same time. Each party numbers the bytes, usually with a different starting byte number. The sequence number in each direction shows the number of the first byte carried by the segment. Each party also uses an acknowledgment number to confirm the bytes it has received. However, the acknowledgment number defines the number of the next byte that the party expects to receive. In addition, the acknowledgment number is cumulative, which means that the party takes the number of the last byte that it has received, safe and sound, adds 1 to it, and announces this sum as the acknowledgment number. The term *cumulative* here means that if a party uses 5,643 as an acknowledgment number, it has received all bytes from the beginning up to 5,642. Note that this does not mean that the party has received 5,642 bytes because the first byte number does not have to start from 0.

The value of the acknowledgment field in a segment defines the number of the next byte a party expects to receive. The acknowledgment number is cumulative.

d. **Explain half-close of TCP.**

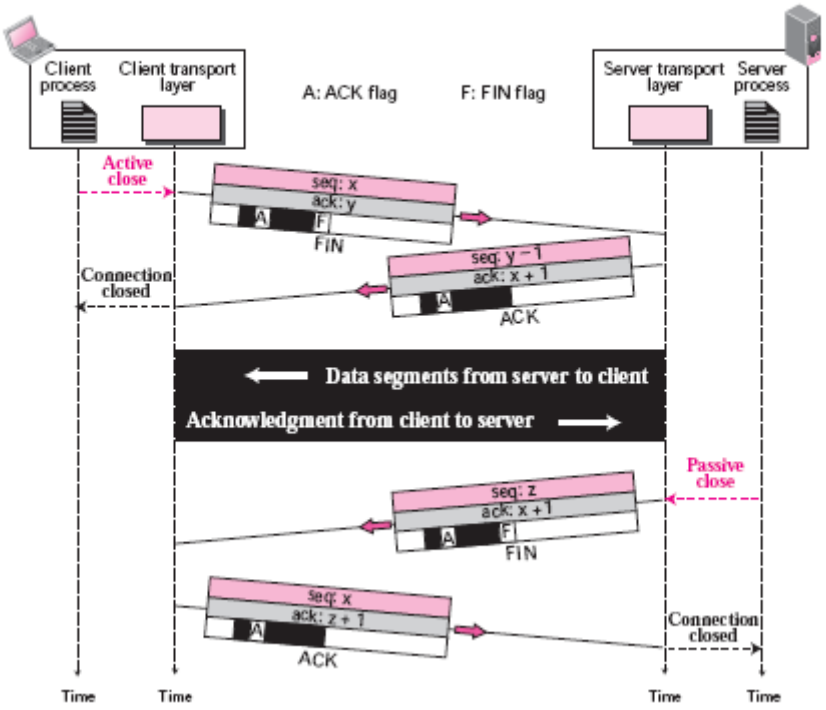
Half-Close

In TCP, one end can stop sending data while still receiving data. This is called a **halfclose.** Either the server or the client can issue a half-close request. It can occur when the server needs all the data before processing can begin. A good example is sorting. When the client sends data to the server to be sorted, the server needs to receive all the data before sorting can start. This means the client, after sending all data, can close the connection in the client-to-server direction. However, the server-to-client direction must remain open to return the sorted data. The server, after receiving the data, still needs time for sorting; its outbound direction must remain open.

The FIN + ACK segment consumes one sequence number if it does not carry data.

Figure 15.12 shows an example of a half-close. The data transfer from the client to the server stops. The client half-closes the connection by sending a FIN segment. The server accepts the half-close by sending the ACK segment. The server, however, can still send data. When the server has sent all of the processed data, it sends a FIN segment, which is acknowledged by an ACK from the client.

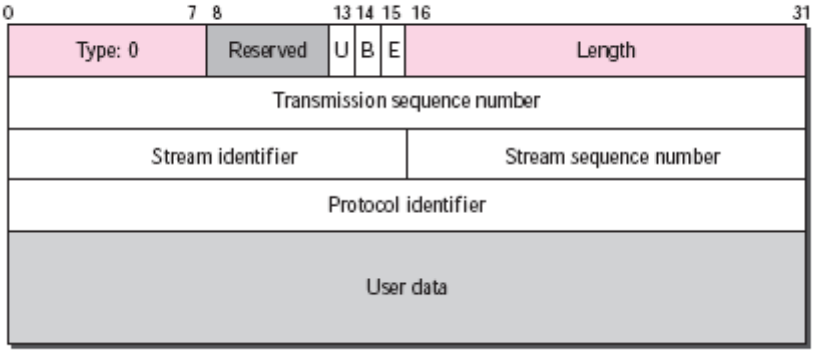
Figure 15.12 *Half-close*



After half closing the connection, data can travel from the server to the client and acknowledgments can travel from the client to the server. The client cannot send any more data to the server. Note the sequence numbers we have used. The second segment (ACK) consumes no sequence number. Although the client has received sequence number *y*- 1 and is expecting *y*, the server sequence number is still *y* - 1. When the connection finally closes, the sequence number of the last ACK segment is still *x*, because no sequence numbers are consumed during data transfer in that direction.

4.	**Attempt *any two* of the following:**	**10**
a.	**Explain format of DATA chunk and INIT chunk.** ***DATA*** The **DATA chunk** carries the user data. A packet may contain zero or more data chunks. Figure 16.9 shows the format of a DATA chunk.	

Figure 16.9 *DATA chunk*



The descriptions of the common fields are the same. The type field has a value of 0. The flag field has 5 reserved bits and 3 defined bits: U, B, and E. The U (unordered) field, when set to 1, signals unordered data (explained later). In this case, the value of the stream sequence number is ignored. The B (beginning) and E (end) bits together define the position of a chunk in a message that is fragmented. When B □□1 and E □□1,
there is no fragmentation (first and last); the whole message is carried in one chunk. When B □□1 and E □□0, it is the first fragment. When B □□0 and E □□1, it is the last fragment. When B □□0 and E □□0, it is a middle fragment (neither the first nor the last).
Note that the value of the length field does not include padding. This value cannot be less than 17 because a DATA chunk must always carry at least one byte of data.

❑**Transmission sequence number (TSN).** This 32-bit field defines the transmission sequence number. It is a sequence number that is initialized in an INIT chunk for one direction and in the INIT ACK chunk for the opposite direction.

❑**Stream identifier (SI).** This 16-bit field defines each stream in an association. All chunks belonging to the same stream in one direction carry the same stream identifier.

❑**Stream sequence number (SSN).** This 16-bit field defines a chunk in a particular stream in one direction.

❑**Protocol identifier.** This 32-bit field can be used by the application program to define the type of data. It is ignored by the SCTP layer.

❑**User data.** This field carries the actual user data. SCTP has some specific rules about the user data field. First, no chunk can carry data belonging to more than one message, but a message can be spread over several data chunks. Second, this field cannot be empty; it must have at least one byte of user data. Third, if the data cannot end at a 32-bit boundary, padding must be added. These padding bytes are not included in the value of the length field.

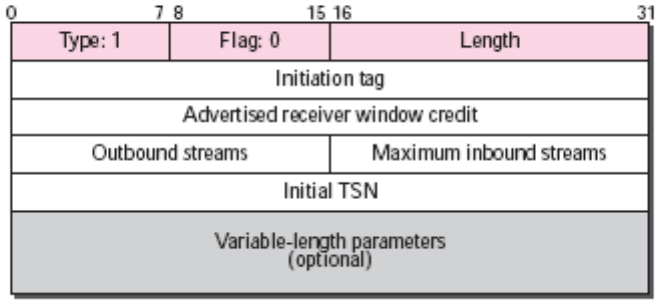
INIT

The **INIT chunk** (initiation chunk) is the first chunk sent by an end point to establish an association. The packet that carries this chunk cannot carry any other control or data chunks. The value of the verification tag for this packet is 0, which means no tag has yet been defined. The format is shown in Figure 16.10.
The three common fields (type, flag, and length) are as before. The value of the type field is 1. The value of the flag field is zero (no flags); and the value of the length field is a minimum of 20 (more if there are optional parameters). The other fields are explained below:

❑**Initiation tag.** This 32-bit field defines the value of the verification tag for packets traveling in the opposite direction. As we mentioned before, all packets have a

verification
tag in the general header; this tag is the same for all packets traveling in one direction in an association. The value of this tag is determined during

Figure 16.10 *INIT chunk*



association establishment. The end point that initiates the association defines the value of this tag in the initiation tag field. This value is used as the verification tag in the rest of the packets sent from the other direction. For example, when end point A starts an association with end point B, A defines an initiation tag value, say *x*, which is used as the verification tag for all packets sent from B to A. The initiation tag is a random number between 0 and 232 − 1. The value of 0 defines no association and is permitted only by the general header of the INIT chunk.

- **Advertised receiver window credit.** This 32-bit field is used in flow control and defines the initial amount of data in bytes that the sender of the INIT chunk can allow. It is the rwnd value that will be used by the receiver to know how much data to send. Note that, in SCTP, sequence numbers are in terms of chunks.
- **Outbound stream.** This 16-bit field defines the number of streams that the initiator of the association suggests for streams in the outbound direction. It may be reduced by the other end point.
- **Maximum inbound stream.** This 16-bit field defines the maximum number of streams that the initiator of the association can support in the inbound direction. Note that this is a maximum number and cannot be increased by the other end point.
- **Initial TSN.** This 32-bit field initializes the transmission sequence number (TSN) in the outbound direction. Note that each data chunk in an association has to have one TSN. The value of this field is also a random number less than 232.
- **Variable-length parameters.** These optional parameters may be added to the INIT chunk to define the IP address of sending end point, the number of IP addresses the end point can support (multihome), the preservation of the cookie state, the type of addresses, and support of explicit congestion notification (ECN).

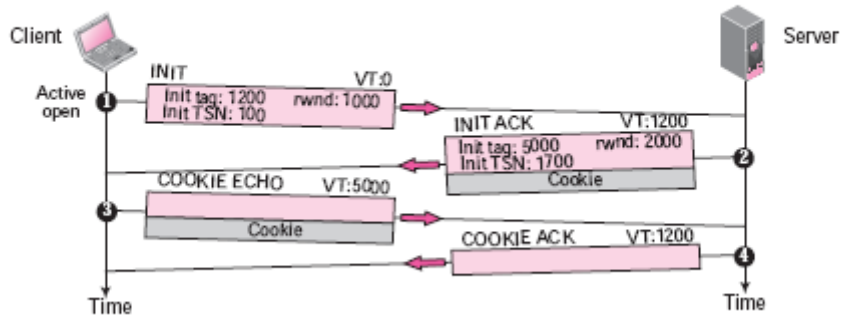
b. **How do you establish an association in SCTP?**

SCTP, like TCP, is a connection-oriented protocol. However, a connection in SCTP is called an *association* to emphasize multihoming.

Association Establishment

Association establishment in SCTP requires a *four-way handshake*. In this procedure, a process, normally a client, wants to establish an association with another process, normally a server, using SCTP as the transport layer protocol. Similar to TCP, the SCTP server needs to be prepared to receive any association (passive open). Association establishment, however, is initiated by the client (active open). SCTP association establishment is shown in Figure 16.19.

Figure 16.19 *Four-way handshaking*



The steps, in a normal situation, are as follows:

1. The client sends the first packet, which contains an INIT chunk. The **verification tag** (VT) of this packet (defined in the general header) is 0 because no verification tag has yet been defined for this direction (client to server).The INIT tag includes an initiation tag to be used for packets from the other direction (server to client). The chunk also defines the initial TSN for this direction and advertises a value for rwnd. The value of rwnd is normally advertised in a SACK chunk; it is done here because SCTP allows the inclusion of a DATA chunk in the third and fourth packets; the server must be aware of the available client buffer size. Note that no other chunks can be sent with the first packet.

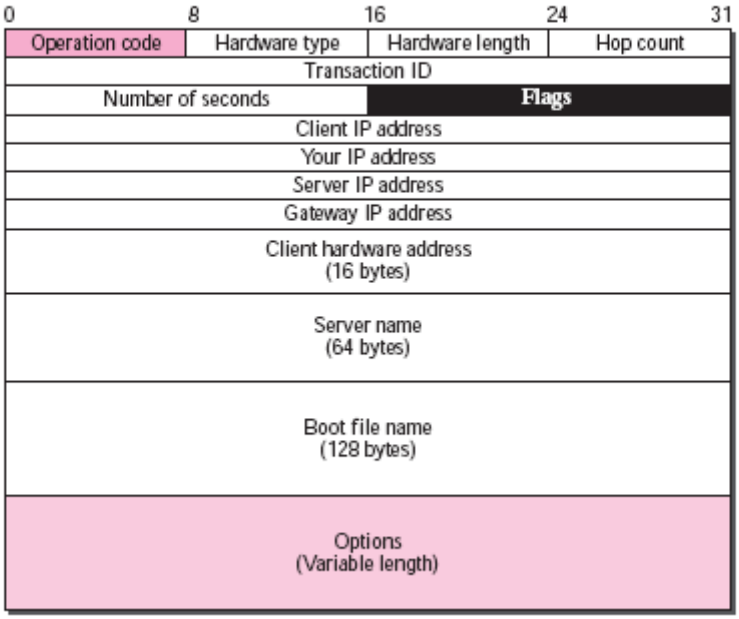
2. The server sends the second packet, which contains an INIT ACK chunk. The verification tag is the value of the initial tag field in the INIT chunk. This chunk initiates the tag to be used in the other direction, defines the initial TSN, for data flow from server to client, and sets the servers' rwnd. The value of rwnd is defined to allow the client to send a DATA chunk with the third packet. The INIT ACK also sends a cookie that defines the state of the server at this moment. We will discuss the use of the cookie shortly.

3. The client sends the third packet, which includes a COOKIE ECHO chunk. This is a very simple chunk that echoes, without change, the cookie sent by the server. SCTP allows the inclusion of data chunks in this packet.

4. The server sends the fourth packet, which includes the COOKIE ACK chunk that acknowledges the receipt of the COOKIE ECHO chunk. SCTP allows the inclusion of data chunks with this packet.

c. **Draw and explain DHCP Packet Format.**

Figure 18.4 *DHCP packet format*



The following briefly describes each field:

❑**Operation code.** This 8-bit field defines the type of DHCP packet: request (1) or reply (2).

❑**Hardware type.** This is an 8-bit field defining the type of physical network. Each type of network has been assigned an integer. For example, for Ethernet the value is 1.

❑**Hardware length.** This is an 8-bit field defining the length of the physical address in bytes. For example, for Ethernet the value is 6.

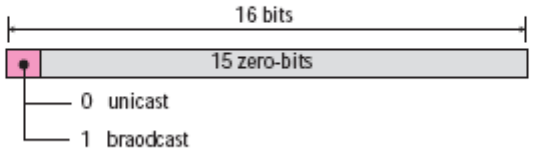
❑**Hop count.** This is an 8-bit field defining the maximum number of hops the packet can travel.

❑**Transaction ID.** This is a 4-byte field carrying an integer. The transaction identification is set by the client and is used to match a reply with the request. The server returns the same value in its reply.

❑**Number of seconds.** This is a 16-bit field that indicates the number of seconds elapsed since the time the client started to boot.

❑**Flag.** This is a 16-bit field in which only the leftmost bit is used and the rest of the bits should be set to 0s. A leftmost bit specifies a forced broadcast reply (instead of unicast) from the server. If the reply were to be unicast to the client, the destination IP address of the IP packet is the address assigned to the client. Since the client does not know its IP address, it may discard the packet. However, if the IP datagram is broadcast, every host will receive and process the broadcast message. Figure 18.5 shows the flag format.

Figure 18.5 *Flag format*



❑**Client IP address.** This is a 4-byte field that contains the client IP address. If the client does not have this information, this field has a value of 0.

❑**Your IP address.** This is a 4-byte field that contains the client IP address. It is

filled by the server (in the reply message) at the request of the client.

❑**Server IP address.** This is a 4-byte field containing the server IP address. It is filled by the server in a reply message.

❑**Gateway IP address.** This is a 4-byte field containing the IP address of a router. It is filled by the server in a reply message.

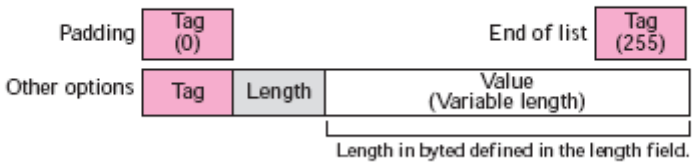
❑**Client hardware address.** This is the physical address of the client. Although the server can retrieve this address from the frame sent by the client, it is more efficient if the address is supplied explicitly by the client in the request message.

❑**Server name.** This is a 64-byte field that is optionally filled by the server in a reply packet. It contains a null-terminated string consisting of the domain name of the server. If the server does not want to fill this field with data, the server must fill it with all 0s.

❑**Boot filename.** This is a 128-byte field that can be optionally filled by the server in a reply packet. It contains a null-terminated string consisting of the full pathname of the boot file. The client can use this path to retrieve other booting information. If the server does not want to fill this field with data, the server must fill it with all 0s.

❑**Options.** This is a 64-byte field with a dual purpose. It can carry either additional information (such as the network mask or default router address) or some specific vendor information. The field is used only in a reply message. The server uses a number, called a **magic cookie,**in the format of an IP address with the value of 99.130.83.99. When the client finishes reading the message, it looks for this magic cookie. If present, the next 60 bytes are options. An option is composed of three fields: a 1-byte tag field, a 1-byte length field, and a variable-length value field. The length field defines the length of the value field, not the whole option. See Figure 18.6.

Figure 18.6 *Option format*

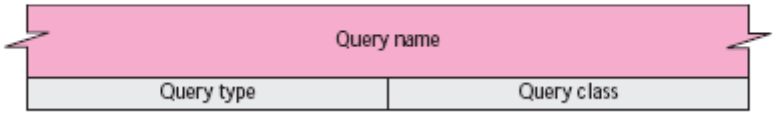


d. **<u>Explain types of records in DNS</u>**.

Question Record

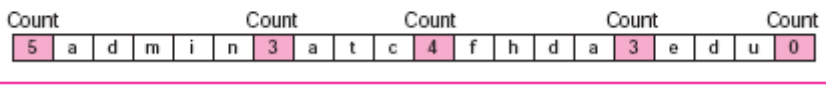
A **question record** is used by the client to get information from a server. This contains the domain name. Figure 19.17 shows the format of a question record. The list below describes question record fields.

Figure 19.17 *Question record format*



❑**Query name.** This is a variable-length field containing a domain name (see Figure 19.18). The count field refers to the number of characters in each section.

Figure 19.18 *Query name format*



❑**Query type.** This is a 16-bit field defining the type of query. Table 19.3 shows

some of the types commonly used. The last two can only be used in a query.

Table 19.3 *Types*

Type	*Mnemonic*	*Description*
1	A	**Address.** A 32-bit IPv4 address. It converts a domain name to an address.
2	NS	**Name server.** It identifies the authoritative servers for a zone.
5	CNAME	**Canonical name.** It defines an alias for the official name of a host.
6	SOA	**Start of authority.** It marks the beginning of a zone.
11	WKS	**Well-known services.** It defines the network services that a host provides.
12	PTR	**Pointer.** It is used to convert an IP address to a domain name.
13	HINFO	**Host information.** It defines the hardware and operating system.
15	MX	**Mail exchange.** It redirects mail to a mail server.
28	AAAA	**Address.** An IPv6 address (see Chapter 26).
252	AXFR	A request for the transfer of the entire zone.
255	ANY	A request for all records.

❑**Query class.** This is a 16-bit field defining the specific protocol using DNS. Table 19.4 shows the current values. In this text we are interested only in class 1 (the Internet).

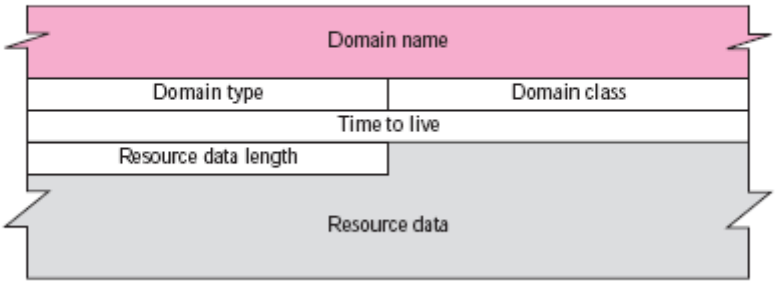
Table 19.4 *Classes*

Class	*Mnemonic*	*Description*
1	IN	Internet
2	CSNET	CSNET network (obsolete)
3	CS	The COAS network
4	HS	The Hesiod server developed by MIT

Resource Record

Each domain name (each node on the tree) is associated with a record called the **resource record.** The server database consists of resource records. Resource records are also what is returned by the server to the client. Figure 19.19 shows the format of a resource record.

Figure 19.19 *Resource record format*



❑**Domain name.** This is a variable-length field containing the domain name. It is a duplication of the domain name in the question record. Since DNS requires the use of compression everywhere a name is repeated, this field is a pointer offset to the corresponding domain name field in the question record. See Section 19.7 on Compression.

❑**Domain type.** This field is the same as the query type field in the question record except the last two types are not allowed. Refer to Table 19.3 for more information.

❑**Domain class.** This field is the same as the query class field in the question record (see Table 19.4).

❑**Time-to-live.** This is a 32-bit field that defines the number of seconds the answer is valid. The receiver can cache the answer for this period of time. A zero value means that the resource record is used only in a single transaction and is not cached.

❑**Resource data length.** This is a 16-bit field defining the length of the resource data.

❑**Resource data.** This is a variable-length field containing the answer to the query (in the answer section) or the domain name of the authoritative server (in the authoritative section) or additional information (in the additional information section). The format and contents of this field depend on the value of the type field. It can be one of the following:

a. A number. This is written in octets. For example, an IPv4 address is a 4-octet integer and an IPv6 address is a 16-octet integer.

b. A domain name. Domain names are expressed as a sequence of labels. Each label is preceded by a 1-byte length field that defines the number of characters in the label. Since every domain name ends with the null label, the last byte of every domain name is the length field with the value of 0. To distinguish between a length field and an offset pointer (as we will discuss later), the two high-order bits of a length field are always zero (00). This will not create a problem because the length of a label cannot be more than 63, which is a maximum of 6 bits (111111).

c. An offset pointer. Domain names can be replaced with an offset pointer. An offset pointer is a 2-byte field with each of the 2 high-order bits set to 1 (11).

d. A character string. A character string is represented by a 1-byte length field followed by the number of characters defined in the length field. The length field is not restricted like the domain name length field. The character string can be as long as 255 characters (including the length field).

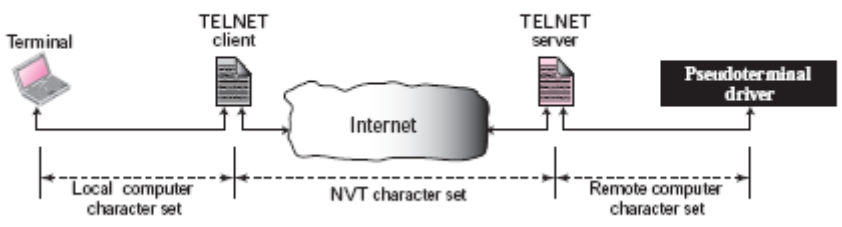
5.	**Attempt *any two* of the following:**	**10**
a.	**Write a note NVT.**	

Network Virtual Terminal (NVT)

The mechanism to access a remote computer is complex. This is because every computer and its operating system accepts a special combination of characters as tokens. For example, the end-of-file token in a computer running the DOS operating system is Ctrl+z, while the UNIX operating system recognizes Ctrl+d.

We are dealing with heterogeneous systems. If we want to access any remote computer in the world, we must first know what type of computer we will be connected to, and we must also install the specific terminal emulator used by that computer. TELNET solves this problem by defining a universal interface called the **Network Virtual Terminal (NVT)** character set. Via this interface, the client TELNET translates characters (data or commands) that come from the local terminal into NVT form and delivers them to the network. The server TELNET, on the other hand, translates data and commands from NVT form into the form acceptable by the remote computer. For an illustration of this concept, see Figure 20.3.

Figure 20.3 *Concept of NVT*



NVT Character Set

NVT uses two sets of characters, one for data and one for control. Both are 8-bit bytes (Figure 20.4).

Figure 20.4 *Format of data and control characters*

Data Characters For data, NVT normally uses what is called NVT ASCII. This is an 8-bit character set in which the seven lowest order bits are the same as US ASCII and the highest order bit is 0 (see Figure 20.4). Although it is possible to send an 8-bit ASCII (with the highest order bit set to be 0 or 1), this must first be agreed upon between the client and the server using option negotiation.

Control Characters

To send **control characters** between computers (from client to server or vice versa), NVT uses an 8-bit character set in which the highest order bit is set to 1 (see Figure 20.4). Table 20.1 lists some of the control characters and their meanings.
Later we will categorize these control characters on the basis of their functionalities.

Table 20.1 *Some NVT control characters*

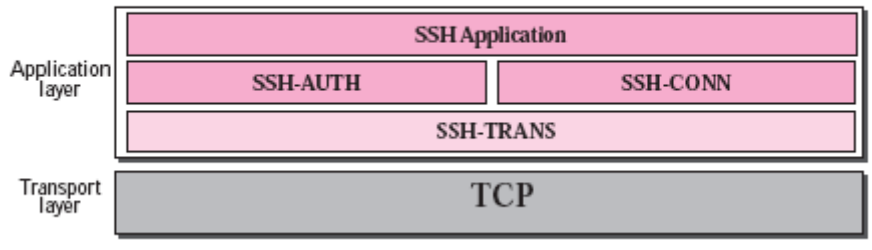
Character	*Decimal*	*Binary*	*Meaning*
EOF	236	11101100	End of file
EOR	239	11101111	End of record
SE	240	11110000	Suboption end
NOP	241	11110001	No operation
DM	242	11110010	Data mark
BRK	243	11110011	Break
IP	244	11110100	Interrupt process
AO	245	11110101	Abort output
AYT	246	11110110	Are you there?
EC	247	11110111	Erase character
EL	248	11111000	Erase line
GA	249	11111001	Go ahead
SB	250	11111010	Suboption begin
WILL	251	11111011	Agreement to enable option
WONT	252	11111100	Refusal to enable option
DO	253	11111101	Approval to option request
DONT	254	11111110	Denial of option request
IAC	255	11111111	Interpret (the next character) as control

b. **Explain in detail components of SSH.**

Components

SSH is a proposed application-layer protocol with four components, as shown in Figure 20.17.

Figure 20.17 *Components of SSH*



SSH Transport-Layer Protocol (SSH-TRANS)

Since TCP is not a secured transport layer protocol, SSH first uses a protocol that creates a secured channel on the top of TCP. This new layer is an independent protocol referred to as SSH-TRANS. When the software implementing this protocol is called, the client and server first use the TCP protocol to establish an insecure proconnection.

Then they exchange several security parameters to establish a secure channel on the top of the TCP. We discuss network security in Chapter 29, but we briefly list the services provided by this protocol:

1. Privacy or confidentiality of the message exchanged.

2. Data integrity, which means that it is guaranteed that the messages exchanged between the client and server are not changed by an intruder.

3. Server authentication, which means that the client is now sure that the server is the one that it claims to be.

4. Compression of the messages that improve the efficiency of the system and makes attack more difficult.

SSH Authentication Protocol (SSH-AUTH)

After a secure channel is established between the client and the server and the server is authenticated for the client, SSH can call another software that can authenticate the client for the server.

SSH Connection Protocol (SSH-CONN)

After the secured channel is established and both server and client are authenticated for each other, SHH can call a piece of software that implements the third protocol, SSHCONN.

One of the services provided by the SSH-CONN protocol is to do multiplexing. SSH-CONN takes the secure channel established by the two previous protocols and lets the client create multiple logical channels over it.

SSH Applications

After the connection phase is completed, SSH allows several application programs to use the connection. Each application can create a logical channel as described above and then benefit from the secured connection. In other words, remote login is one of the services that can use the SSH-CONN protocols; other applications, such as a file transfer application can use one of the logical channels for this purpose. In the next chapter, we show how SSH can be used for secure file transfer.

c. **Which are the two approaches that FTP client and server uses to communicate with each other?**

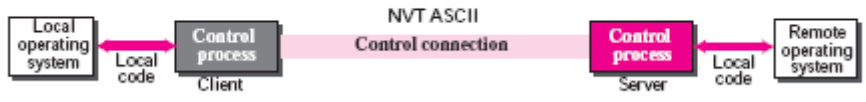
FTP has two different approaches, one for the control connection and one for the data connection. We will study each approach separately.

Communication over Control Connection

FTP uses the same approach as TELNET or SMTP to communicate across the control connection. It uses the NVT ASCII character set (see Figure 21.4). Communication is achieved through commands and responses. This simple method is adequate for the control connection because we send one command (or response) at a time. Each command

or response is only one short line so we need not worry about file format or file

Figure 21.4 *Using the control connection*



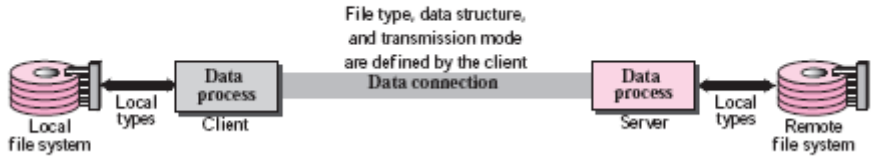
structure. Each line is terminated with a two-character (carriage return and line feed) end-of-line token.

Communication over Data Connection

The purpose and implementation of the data connection are different from that of the control connection. We want to transfer files through the data connection. The client must define the type of file to be transferred, the structure of the data, and the transmission mode. Before sending the file through the data connection, we prepare for transmission through the control connection. The heterogeneity problem is resolved by defining three attributes of communication: file type, data structure, and transmission

mode (see Figure 21.5).

Figure 21.5 *Using the data connection*



d. **Explain Persistence and non-persistence connection of HTTP**

Nonpersistent Connection

In a **nonpersistent connection,** one TCP connection is made for each request/response. The following lists the steps in this strategy:

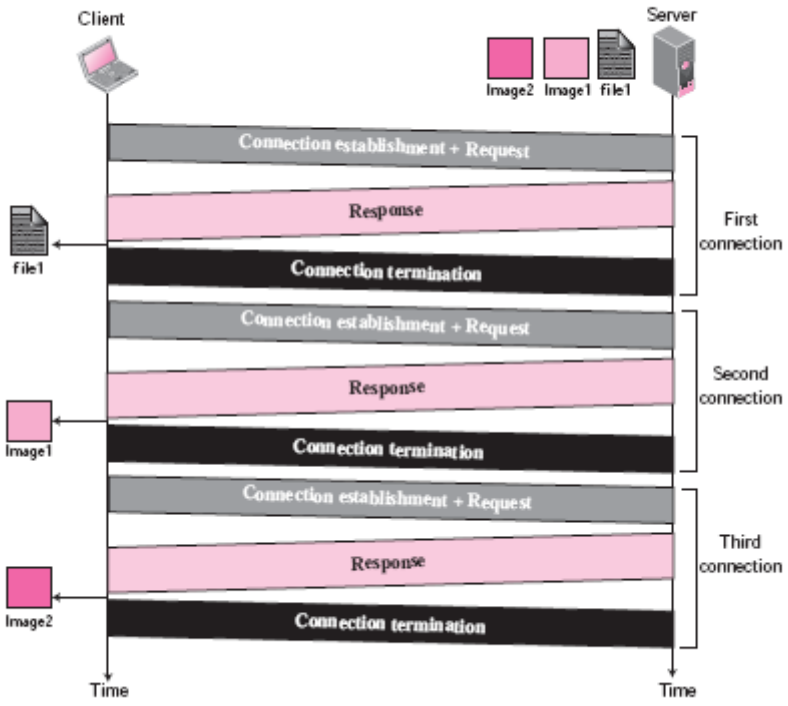
1. The client opens a TCP connection and sends a request.
2. The server sends the response and closes the connection.
3. The client reads the data until it encounters an end-of-file marker; it then closes the connection.

In this strategy, if a file contains link 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 and requires a slow start procedure each time a connection is opened.

Example 22.8

Figure 22.15 shows an example of a nonpersistent connection. The client needs to access a file
that contains two links to images. The text file and images are located on the same server.

Figure 22.15 *Example 22.8*



Persistent Connection

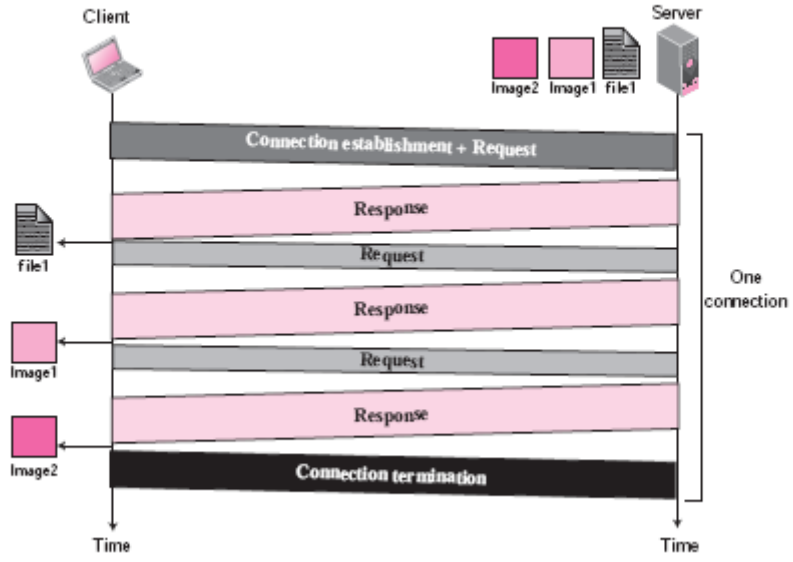
HTTP version 1.1 specifies a **persistent connection** by default. In a persistent connection,
the server leaves the connection open for more requests after sending a response.
The server can close the connection at the request of a client or if a time-out has been

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.

Example 22.9

Figure 22.16 shows the same scenario as Example 22.8, but using persistent connection.

Figure 22.16 *Example 22.9*



6.	**Attempt *any two* of the following:**	**10**
a.	**Explain the phases of mail transfer**.	

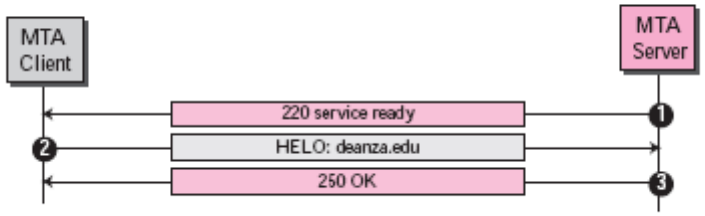
Mail Transfer Phases

The process of transferring a mail message occurs in three phases: **connection establishment,**
mail transfer, and **connection termination.**

Connection Establishment

After a client has made a TCP connection to the well-known port 25, the SMTP server starts the connection phase. This phase involves the following three steps, which are illustrated in Figure 23.10.

Figure 23.10 *Connection establishment*



1. The server sends code 220 (service ready) to tell the client that it is ready to receive mail. If the server is not ready, it sends code 421 (service not available).
2. The client sends the HELO message to identify itself using its domain name address. This step is necessary to inform the server of the domain name of the client. Remember that during TCP connection establishment, the sender and receiver know each other through their IP addresses.
3. The server responds with code 250 (request command completed) or some other code depending on the situation.

Message Transfer

After connection has been established between the SMTP client and server, a single message between a sender and one or more recipients can be exchanged. This phase involves eight steps. Steps 3 and 4 are repeated if there is more than one recipient (see Figure 23.11).

1. The client sends the MAIL FROM message to introduce the sender of the message. It includes the mail address of the sender (mailbox and the domain name). This step is needed to give the server the return mail address for returning errors and reporting messages.

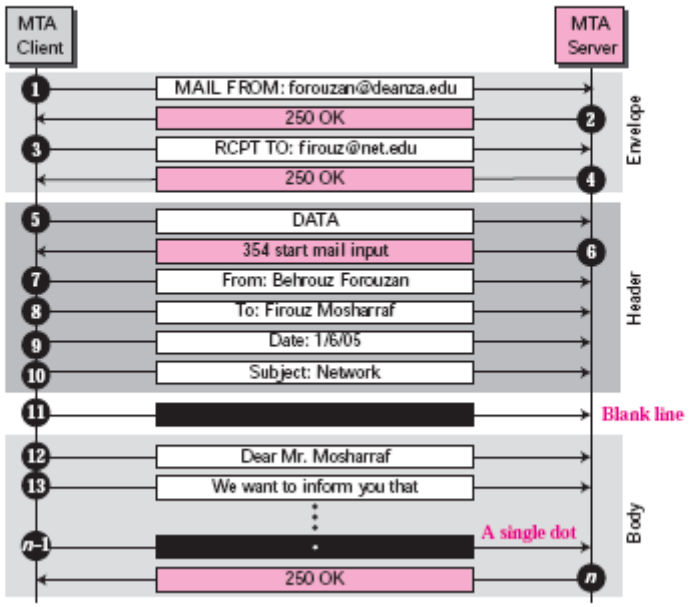
2. The server responds with code 250 or some other appropriate code.

3. The client sends the RCPT TO (recipient) message, which includes the mail address of the recipient.

4. The server responds with code 250 or some other appropriate code.

5. The client sends the DATA message to initialize the message transfer.

Figure 23.11 *Message transfer*



6. The server responds with code 354 (start mail input) or some other appropriate message.

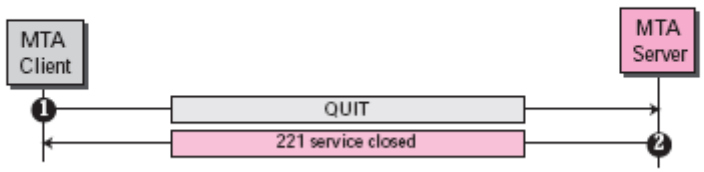
7. The client sends the contents of the message in consecutive lines. Each line is terminated by a two-character end-of-line token (carriage return and line feed). The message is terminated by a line containing just one period.

8. The server responds with code 250 (OK) or some other appropriate code.

Connection Termination

After the message is transferred successfully, the client terminates the connection. This phase involves two steps (see Figure 23.12).

Figure 23.12 *Connection termination*



1. The client sends the QUIT command.

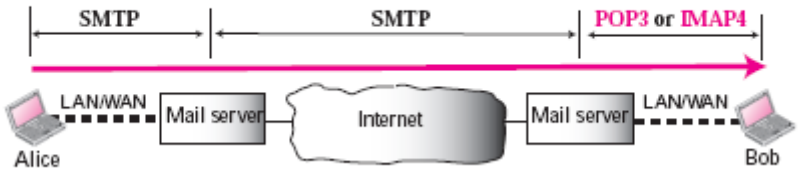
2. The server responds with code 221 or some other appropriate code.

After the connection termination phase, the TCP connection must be closed.

b. **What is the role of Message Access Agent? How does POP3 work?**

The first and the second stages of mail delivery use SMTP. However, SMTP is not involved in the third stage because SMTP is a *push* protocol; it pushes the message from the client to the server. In other words, the direction of the bulk data (messages) is from the client to the server. On the other hand, the third stage needs a *pull* protocol; the client must pull messages from the server. The direction of the bulk data are from the server to the client. The third stage uses a message access agent.
Currently two message access protocols are available: Post Office Protocol, version 3 (POP3) and Internet Mail Access Protocol, version 4 (IMAP4). Figure 23.13 shows the position of these two protocols in the most common situation (fourth scenario).

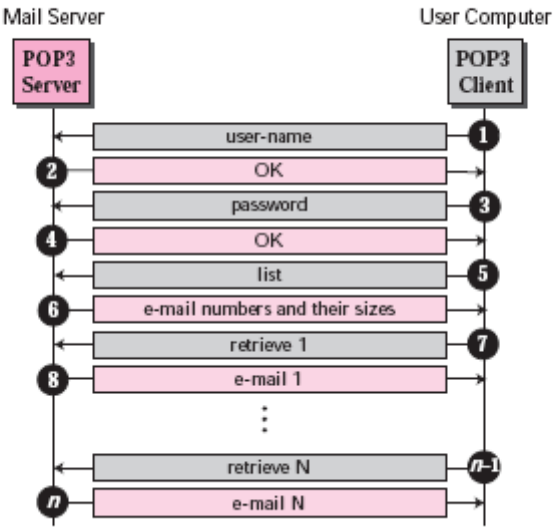
Figure 23.13 *POP3 and IMAP4*



POP3
Post Office Protocol, version 3 (POP3) is simple and limited in functionality. The client POP3 software is installed on the recipient computer; the server POP3 software is installed on the mail server.

Mail access starts with the client when the user needs to download its e-mail from the mailbox on the mail server. The client opens a connection to the server on TCP port 110. It then sends its user name and password to access the mailbox. The user can then list and retrieve the mail messages, one by one. Figure 23.14 shows an example of downloading using POP3.

Figure 23.14 *POP3*



POP3 has two modes: the delete mode and the keep mode. In the delete mode, the mail is deleted from the mailbox after each retrieval. In the keep mode, the mail remains in the mailbox after retrieval. The delete mode is normally used when the user is working at her permanent computer and can save and organize the received mail after reading or replying. The keep mode is normally used when the user accesses her mail away from her primary computer (e.g., a laptop). The mail is read but kept in the system for later retrieval and organizing.

c. **What is the concept of MIME? Explain its format.**

Multipurpose Internet Mail Extensions (MIME) is a supplementary protocol that allows non-ASCII data to be sent through e-mail. MIME transforms non-ASCII data at

the sender site to NVT ASCII data and delivers it to the client MTA to be sent through the Internet. The message at the receiving site is transformed back to the original data. We can think of MIME as a set of software functions that transforms non-ASCII data to ASCII data and vice versa, as shown in Figure 23.15.

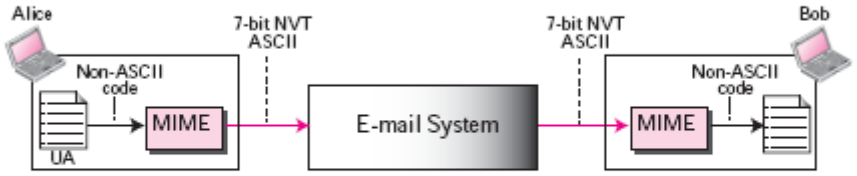
MIME Headers

MIME defines five headers that can be added to the original e-mail header section to define the transformation parameters:

1. MIME-Version

2. Content-Type

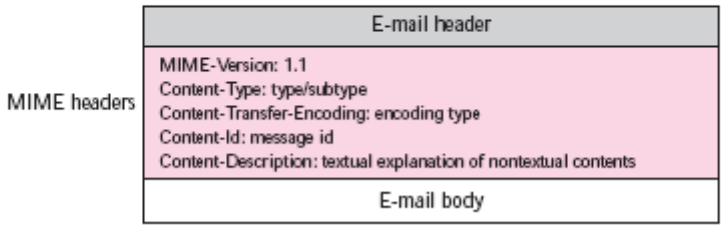
Figure 23.15 *MIME*



3. Content-Transfer-Encoding
4. Content-Id
5. Content-Description

Figure 23.16 shows the MIME headers. We will describe each header in detail.

Figure 23.16 *MIME header*



MIME-Version

This header defines the version of MIME used. The current version is 1.1.

Content-Type

This header defines the type of data used in the body of the message. The content type and the content subtype are separated by a slash. Depending on the subtype, the header may contain other parameters. MIME allows seven different types of data, listed in Table 23.3.

Table 23.3 *Data Types and Subtypes in MIME*

Type	*Subtype*	*Description*
Text	Plain	Unformatted
	HTML	HTML format (see Appendix E)
Multipart	Mixed	Body contains ordered parts of different data types
	Parallel	Same as above, but no order
	Digest	Similar to Mixed, but the default is message/RFC822
	Alternative	Parts are different versions of the same message
Message	RFC822	Body is an encapsulated message
	Partial	Body is a fragment of a bigger message
	External-Body	Body is a reference to another message
Image	JPEG	Image is in JPEG format
	GIF	Image is in GIF format
Video	MPEG	Video is in MPEG format
Audio	Basic	Single channel encoding of voice at 8 KHz
Application	PostScript	Adobe PostScript
	Octet-stream	General binary data (eight-bit bytes)

Content-Transfer-Encoding

This header defines the method used to encode the messages into 0s and 1s for

transport:
The five types of encoding methods are listed in Table 23.4.

Table 23.4 *Content-Transfer-Encoding*

Type	*Description*
7bit	NVT ASCII characters and short lines
8bit	Non-ASCII characters and short lines
Binary	Non-ASCII characters with unlimited-length lines
Base64	6-bit blocks of data are encoded into 8-bit ASCII characters
Quoted-printable	Non-ASCII characters are encoded as an equal sign plus an ASCII code

Content-Id
This header uniquely identifies the whole message in a multiple message environment.
Content-Description
This header defines whether the body is image, audio, or video.

d. **How do you compress an audio?**

Audio Compression
Audio compression can be used for speech or music. For speech, we need to compress a 64-kHz digitized signal; for music, we need to compress a 1.411-MHz signal. Two categories of techniques are used for audio compression: predictive encoding and perceptual encoding.

Predictive Encoding In **predictive encoding,** the differences between the samples are encoded instead of encoding all the sampled values. This type of compression is normally used for speech. Several standards have been defined such as GSM (13 kbps), G.729 (8 kbps), and G.723.3 (6.4 or 5.3 kbps). Detailed discussions of these techniques are beyond the scope of this book.

Perceptual Encoding: MP3
The most common compression technique that is used to create CD-quality audio is based on the **perceptual encoding** technique. As we mentioned before, this type of audio needs at least 1.411 Mbps; this cannot be sent over the Internet without compression.

MP3
(MPEG audio layer 3), a part of the MPEG standard (discussed in the video compression section), uses this technique.Perceptual encoding is based on the science of psychoacoustics, which is the study of how people perceive sound. The idea is based on some flaws in our auditory system: Some sounds can mask other sounds. Masking can happen in frequency and time.

In **frequency masking,** a loud sound in a frequency range can partially or totally mask a softer sound in another frequency range. For example, we cannot hear what our dance partner says in a room where a loud heavy metal band is performing.

In **temporal masking,** a loud sound can numb our ears for a short time even after the sound has stopped.
MP3 uses these two phenomena, frequency and temporal masking, to compress audio signals. The technique analyzes and divides the spectrum into several groups. Zero bits are allocated to the frequency ranges that are totally masked. A small number of bits are allocated to the frequency ranges that are partially masked. A larger number of bits are allocated to the frequency ranges that are not masked.
MP3 produces three data rates: 96 kbps, 128 kbps, and 160 kbps. The rate is based on the range of the frequencies in the original analog audio.

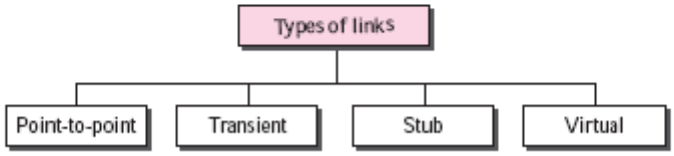
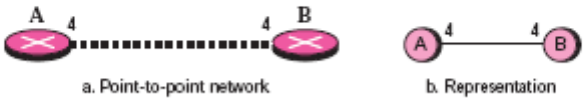
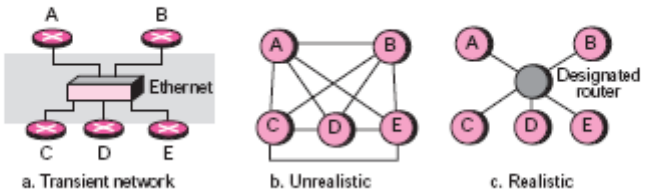
7.	**Attempt *any three* of the following:**	**15**
a.	**Comparison between IPv4 and IPv6 options.** 1. The no-operation and end-of-option options in IPv4 are replaced by Pad1 and PadN options in IPv6. 2. The record route option is not implemented in IPv6 because it was not used. 3. The timestamp option is not implemented because it was not used. 4. The source route option is called the source route extension header in IPv6. 5. The fragmentation fields in the base header section of IPv4 have moved to the fragmentation 6. extension header in IPv6. 7. The authentication extension header is new in IPv6. 8. The encrypted security payload extension header is new in IPv6.	
b.	**What are the types of links in OSPF? Explain each in detail.** **Types of Links** In OSPF terminology, a connection is called a *link*. Four types of links have been defined: point-to-point, transient, stub, and virtual (see Figure 11.22). **Figure 11.22** *Types of links*  ***Point-to-Point Link*** A **point-to-point link** connects two routers without any other host or router in between. In other words, the purpose of the link (network) is just to connect the two routers. An example of this type of link is two routers connected by a telephone line or a T-line. There is no need to assign a network address to this type of link. Graphically, the routers are represented by nodes, and the link is represented by a bidirectional edge connecting the nodes. The metrics, which are usually the same, are shown at the two ends, one for each direction. In other words, each router has only one neighbor at the other side of the link (see Figure 11.23). **Figure 11.23** *Point-to-point link*  ***Transient Link*** A **transient link** is a network with several routers attached to it. The data can enter through any of the routers and leave through any router. All LANs and some WANs with two or more routers are of this type. In this case, each router has many neighbors. For example, consider the Ethernet in Figure 11.24a. Router A has routers B, C, D, and E as neighbors. Router B has routers A, C, D, and E as neighbors. If we want to show the neighborhood relationship in this situation, we have the graph shown in Figure 11.24b.	

Figure 11.24 *Transient link*



This is neither efficient nor realistic. It is not efficient because each router needs to advertise the neighborhood to four other routers, for a total of 20 advertisements. It is not realistic, because there is no single network (link) between each pair of routers; there is only one network that serves as a crossroad between all five routers.

To show that each router is connected to every other router through one single network, the network itself is represented by a node. However, because a network is not a machine, it cannot function as a router. One of the routers in the network takes this responsibility. It is assigned a dual purpose; it is a true router and a designated router. We can use the topology shown in Figure 11.24c to show the connections of a transient network.

Now each router has only one neighbor, the designated router (network). On the other hand, the designated router (the network) has five neighbors. We see that the number of neighbor announcements is reduced from 20 to 10. Still, the link is represented as a bidirectional edge between the nodes. However, while there is a metric from each node to the designated router, there is no metric from the designated router to any other node. The reason is that the designated router represents the network. We can only assign a cost to a packet that is passing through the network. We cannot charge for this twice. When a packet enters a network, we assign a cost; when a packet leaves the network to go to the router, there is no charge.

Stub Link

A **stub link** is a network that is connected to only one router. The data packets enter the network through this single router and leave the network through this same router. This is a special case of the transient network. We can show this situation using the router as a node and using the designated router for the network. However, the link is only one directional, from the router to the network (see Figure 11.25).

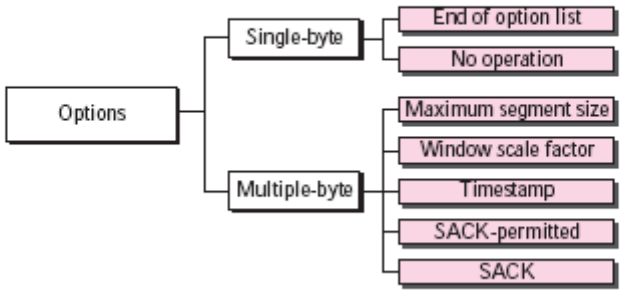
Figure 11.25 *Stub link*

Virtual Link

When the link between two routers is broken, the administration may create a **virtual link** between them using a longer path that probably goes through several routers.

c. **What are the options available in TCP? Explain each with its packet format.**

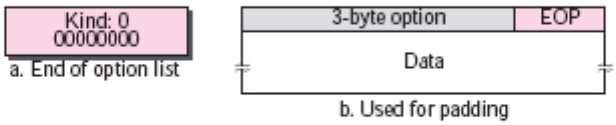
Figure 15.41 *Options*



End of Option (EOP)

The **end-of-option (EOP) option** is a 1-byte option used for padding at the end of the option section. It can only be used as the last option. Only one occurrence of this option is allowed. After this option, the receiver looks for the payload data. Figure 15.42 shows an example. A 3-byte option is used after the header; the data section follows this option. One EOP option is inserted to align the data with the boundary of the next word.

Figure 15.42 *End-of-option*



EOP can be used only once.

The EOP option imparts two pieces of information to the destination:

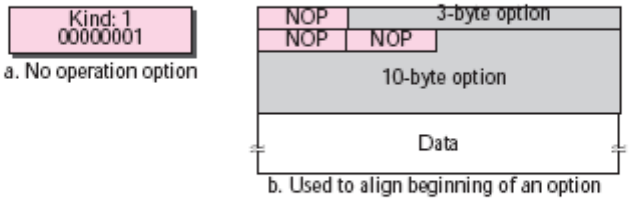
1. There are no more options in the header.

2. Data from the application program starts at the beginning of the next 32-bit word.

No Operation (NOP)

The **no-operation** (**NOP**) **option** is also a 1-byte option used as a filler. However, it normally comes before another option to help align it in a four-word slot. For example, in Figure 15.43 it is used to align one 3-byte option such as the window scale factor and one 10-byte option such as the timestamp.

Figure 15.43 *No-operation option*



NOP can be used more than once.

Maximum Segment Size (MSS)

The **maximum-segment-size option** defines the size of the biggest unit of data that can be received by the destination of the TCP segment. In spite of its name, it defines the maximum size of the data, not the maximum size of the segment. Since the field is 16 bits long, the value can be 0 to 65,535 bytes. Figure 15.44 shows the format of this option.

Figure 15.44 *Maximum-segment-size option*

MSS is determined during connection establishment. Each party defines the MSS for the segments it will receive during the connection. If a party does not define this, the default values is 536 bytes.

The value of MSS is determined during connection establishment and does not change during the connection.

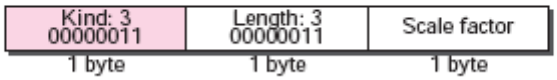
Window Scale Factor

The window size field in the header defines the size of the sliding window. This field is 16 bits long, which means that the window can range from 0 to 65,535 bytes. Although this seems like a very large window size, it still may not be sufficient, especially if the data are traveling through a *long fat pipe*, a long channel with a wide bandwidth. To increase the window size, a **window scale factor** is used. The new window size is found by first raising 2 to the number specified in the window scale factor. Then this result is multiplied by the value of the window size in the header.

$$\text{New window size} = \text{window size defined in the header} \times 2^{\text{window scale factor}}$$

Figure 15.45 shows the format of the window-scale-factor option.

Figure 15.45 *Window-scale-factor option*



The scale factor is sometimes called the *shift count* because multiplying a number by a power of 2 is the same as a left shift in a bitwise operation. In other words, the actual value of the window size can be determined by taking the value of the window size advertisement in the packet and shifting it to the left in the amount of the window scale factor. For example, suppose the value of the window scale factor is 3. An end point receives an acknowledgment in which the window size is advertised as 32,768. The size of window this end can use is 32,768 . 23 or 262,144 bytes. The same value can be obtained if we shift the number 32,768 three bits to the left.

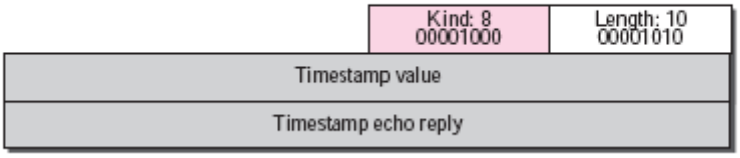
Although the scale factor could be as large as 255, the largest value allowed by TCP/IP is 14, which means that the maximum window size is 216 . 214 = 230, which is less than the maximum value for the sequence number. Note that the size of the window cannot be greater than the maximum value of the sequence number. The window scale factor can also be determined only during the connection establishment phase. During data transfer, the size of the window (specified in the header) may be changed, but it must be multiplied by the same window scale factor.

Note that one end may set the value of the window scale factor to 0, which means that although it supports this option, it does not want to use it for this connection.

Timestamp

This is a 10-byte option with the format shown in Figure 15.46. Note that the end with the active open announces a timestamp in the connection request segment (SYN segment). If it receives a timestamp in the next segment (SYN + ACK) from the other end, it is allowed to use the timestamp; otherwise, it does not use it any more. The **timestamp option** has two applications: it measures the round-trip time and prevents wraparound sequence numbers.

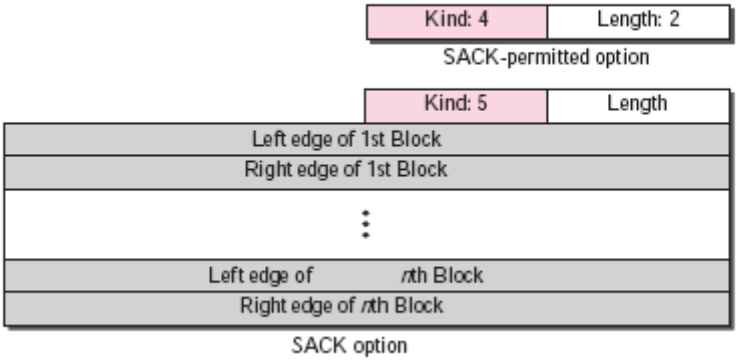
Figure 15.46 *Timestamp option*



SACK-Permitted and SACK Options

As we discussed before, the acknowledgment field in the TCP segment is designed as an accumulative acknowledgment, which means it reports the receipt of the last consecutive byte: it does not report the bytes that have arrived out of order. It is also silent about duplicate segments. This may have a negative effect on TCP's performance. If some packets are lost or dropped, the sender must wait until a time-out and then send all packets that have not been acknowledged. The receiver may receive duplicate packets. To improve performance, selective acknowledgment (SACK) was proposed. Selective acknowledgment allows the sender to have a better idea of which segments are actually lost and which have arrived out of order. The new proposal even includes a list for duplicate packets. The sender can then send only those segments that are really lost. The list of duplicate segments can help the sender find the segments which have been retransmitted by a short time-out. The proposal defines two new options: SACK-permitted and SACK as shown in Figure 15.48.

Figure 15.48 *SACK*



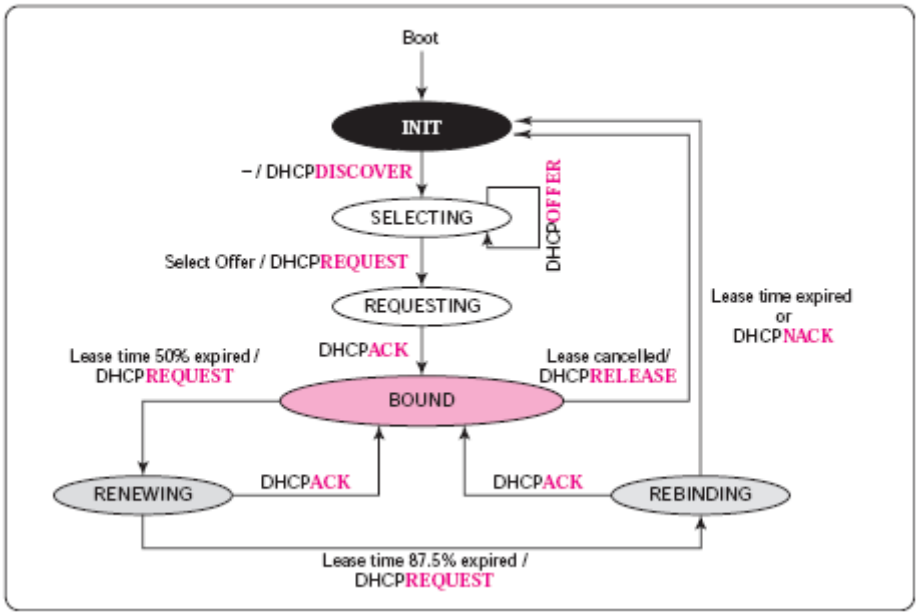
The ***SACK*-permitted option** of two bytes is used only during connection establishment. The host that sends the SYN segment adds this option to show that it can support the SACK option. If the other end, in its SYN + ACK segment, also includes this option, then the two ends can use the SACK option during data transfer. Note that the SACK-permitted option is not allowed during the data transfer phase. The **SACK option,** of variable length, is used during data transfer only if both ends agree (if they have exchanged SACK-permitted options during connection establishment).

The option includes a list for blocks arriving out of order. Each block occupies two 32-bit numbers that define the beginning and the end of the blocks. We will show the use of this option in examples; for the moment, remember that the allowed size of an option in TCP is only 40 bytes. This means that a SACK option cannot define more than 4 blocks. The information for 5 blocks occupies (5 . 2) . 4 + 2 or 42 bytes, which is beyond the available size for the option section in a segment. If the SACK option is used with other options, then the number of blocks may be reduced.

The first block of the SACK option can be used to report the duplicates. This is used only if the implementation allows this feature.

d. **Draw and explain DHCP client state transition diagram.**

Figure 18.8 *DHCP client transition diagram*



INIT State

When the DHCP client first starts, it is in the INIT state (initializing state). The client broadcasts a DHCPDISCOVER message (a request message with the DHCPDISCOVER option), using port 67.

SELECTING State

After sending the DHCPDISCOVER message, the client goes to the selecting state. Those servers that can provide this type of service respond with a DHCPOFFER message.

In these messages, the servers offer an IP address. They can also offer the lease duration. The default is 1 hour. The server that sends a DHCPOFFER locks the offered IP address so that it is not available to any other clients. The client chooses one of the offers and sends a DHCPREQUEST message to the selected server. It then goes to the requesting state. However, if the client receives no DHCPOFFER message, it tries four more times, each with a span of 2 seconds. If there is no reply to any of these DHCPDISCOVERs, the client sleeps for 5 minutes before trying again.

REQUESTING State

The client remains in the requesting state until it receives a DHCPACK message from the server that creates the binding between the client physical address and its IP address. After receipt of the DHCPACK, the client goes to the bound state.

BOUND State

In this state, the client can use the IP address until the lease expires. When 50 percent of the lease period is reached, the client sends another DHCPREQUEST to ask for renewal. It then goes to the renewing state. When in the bound state, the client can also cancel the lease and go to the initializing state.

RENEWING State

The client remains in the renewing state until one of two events happens. It can receive a DHCPACK, which renews the lease agreement. In this case, the client resets its timer and goes back to the bound state. Or, if a DHCPACK is not received, and 87.5 percent of the lease time expires, the client goes to the rebinding state.

REBINDING State

The client remains in the rebinding state until one of three events happens. If the client receives a DHCPNACK or the lease expires, it goes back to the initializing state and tries to get another IP address. If the client receives a DHCPACK, it goes to the bound state and resets the timer.

e. **Describe the architecture of WWW.**

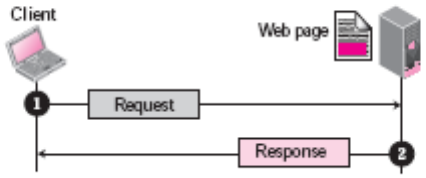
22.1 ARCHITECTURE

The WWW today is a distributed client-server service, in which a client using a browser can access a service using a server. However, the service provided is distributed over many locations called *sites.* Each site holds one or more documents, referred to as Web pages. Each **Web page**, however, can contain some links to other Web pages in the same or other sites. In other words, a Web page can be simple or composite. A simple Web page has no link to other Web pages; a composite Web page has one or more links to other Web pages. Each Web page is a file with a name and address.

Example 22.1

Assume we need to retrieve a Web page that contains the biography of a famous character with some pictures, which are embedded in the page itself. Since the pictures are not stored as separate files, the whole document is a simple Web page. It can be retrieved using one single request/ response transaction, as shown in Figure 22.1.

Figure 22.1 *Example 22.1*

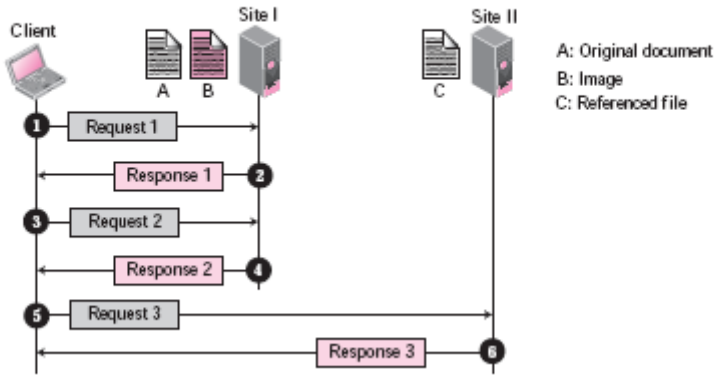


Example 22.2

Now assume we need to retrieve a scientific document that contains one reference to another text file and one reference to a large image. Figure 22.2 shows the situation.

The main document and the image are stored in two separate files in the same site (file A and file B); the referenced text file is stored in another site (file C). Since we are dealing with three different files, we need three transactions if we want to see the whole document. The first transaction (request/response) retrieves a copy of the main document (file A), which has a reference (pointer) to the second and the third files. When a copy of the main document is retrieved and browsed, the user can click on the reference to the image to invoke the second transaction and retrieve a copy of the image (file B). If the user further needs to see the contents of the referenced text file, she can click on its reference (pointer) invoking the third transaction and retrieving a copy of the file C. Note that although files A and B both are stored in site I, they are independent files with different names and addresses. Two transactions are needed to retrieve them.

Figure 22.2 *Example 22.2*



Example 22.3

A very important point we need to remember is that file A, file B, and file C in Example

22.2 are independent Web pages, each with independent names and addresses. Although references to file B or C are included in file A, it does not mean that each of these files cannot be retrieved independently. A second user can retrieve file B with one transaction. A third user can retrieve file C with one transaction.

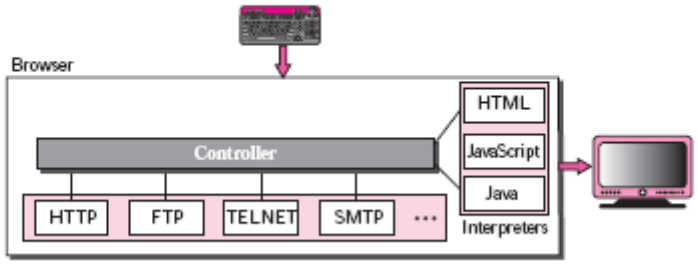
Hypertext and Hypermedia

The three previous examples show the idea of **Hypertext** and **hypermedia**. Hypertext means creating documents that refer to other documents. In a hypertext document, a part of text can be defined as a link to another document. When a hypertext is viewed with a browser, the link can be clicked to retrieve the other document. Hypermedia is a term applied to document that contains links to other textual document or documents containing graphics, video, or audio.

Web Client (Browser)

A variety of vendors offer commercial **browsers** that interpret and display a Web document, and all of them use nearly the same architecture. Each browser usually consists of three parts: a controller, client protocol, and interpreters. (see Figure 22.3).

Figure 22.3 *Browser*



The controller receives input from the keyboard or the mouse and uses the client programs to access the document. After the document has been accessed, the controller uses one of the interpreters to display the document on the screen. The client protocol can be one of the protocols described previously such as FTP, or TELNET, or HTTP (as discussed later in the chapter). The interpreter can be HTML, Java, or JavaScript, depending on the type of document. We discuss the use of these interpreters based on the document type later in the chapter. Some commercial browsers include Internet Explorer, Netscape Navigator, and Firefox.

Web Server

The Web page is stored at the server. Each time a client request arrives, the corresponding document is sent to the client. To improve efficiency, servers normally store requested files in a cache in memory; memory is faster to access than disk. A server can also become more efficient through multithreading or multiprocessing. In this case, a server can answer more than one request at a time. Some popular Web servers include Apache and Microsoft Internet Information Server.

Uniform Resource Locator (URL)

A client that wants to access a Web page needs the file name and the address. To facilitate the access of documents distributed throughout the world, HTTP uses locators. The **uniform resource locator (URL)** is a standard locator for specifying any kind of information on the Internet. The URL defines four things: protocol, host computer, port, and path (see Figure 22.4).

Figure 22.4 *URL*

The *protocol* is the client-server application program used to retrieve the document. Many different protocols can retrieve a document; among them are Gopher, FTP,

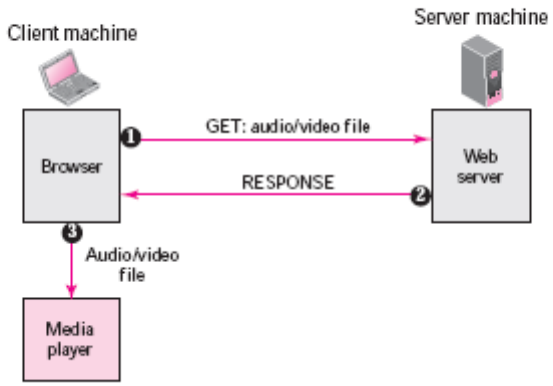
HTTP, News, and TELNET. The most common today is HTTP.
The **host** is the domain name of the computer on which the information is located. Web pages are usually stored in computers, and computers are given domain name aliases that usually begin with the characters "www". This is not mandatory, however, as the host can have any domain name.
The URL can optionally contain the port number of the server. If the *Port* is included, it is inserted between the host and the path, and it is separated from the host by a colon.

Path
is the pathname of the file where the information is located. Note that the path can itself contain slashes that, in the UNIX operating system, separate the directories from the subdirectories and files. In other words, the path defines the complete file name where the document is stored in the directory system.

f. **How do you download a compressed audio/video using a web server and using a media server?**

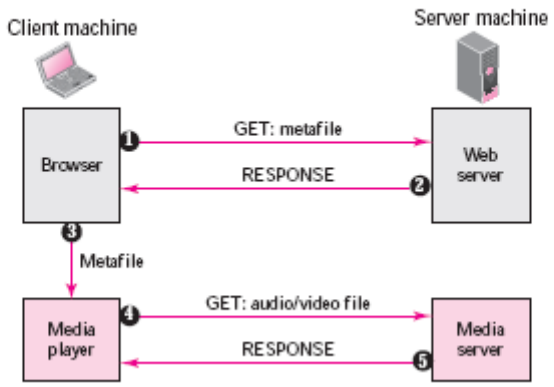
Using a Web Server
A compressed audio/video file can be downloaded as a text file. The client (browser) can use the services of HTTP and send a GET message to download the file. The Web server can send the compressed file to the browser. The browser can then use a help application, normally called a **media player,** to play the file. Figure 25.10 shows this approach.
This approach is very simple and does not involve *streaming*. However, it has a drawback. An audio/video file is usually large even after compression. An audio file may contain tens of megabits, and a video file may contain hundreds of megabits. In this approach, the file needs to download completely before it can be played. Using contemporary data rates, the user needs some seconds or tens of seconds before the file can be played.

Figure 25.10 *Using a Web server*



Using a Media Server
The problem with the second approach is that the browser and the media player both use the services of HTTP. HTTP is designed to run over TCP. This is appropriate for retrieving the metafile, but not for retrieving the audio/video file. The reason is that TCP retransmits a lost or damaged segment, which is counter to the philosophy of streaming. We need to dismiss TCP and its error control; we need to use UDP. However, HTTP, which accesses the Web server, and the Web server itself are designed for TCP; we need another server, a **media server.** Figure 25.12 shows the concept.

Figure 25.12 *Using a media server*



1. The HTTP client accesses the Web server using a GET message.
2. The information about the metafile comes in the response.
3. The metafile is passed to the media player.
4. The media player uses the URL in the metafile to access the media server to download the file. Downloading can take place by any protocol that uses UDP.
5. The media server responds