Internal Assessment Test 2 – November 2025

Sub:	Computer Networks					Sub Code:	BCS502	Branch:	CSE -AIML
Date:	1/11/25	Duration:	90 minutes	Max Marks:	50	Sem/Sec:	V		OBE

		Answer any FIVE FULL Questions	MARKS	CO	RBT
1	a	Illustrate how the architecture of electronic mail (email) operates in a real-world communication scenario.	[10]	3	L3
2	a	Explain MOSPF with an example and suitable diagram.	[10]	3	L2
3	a	Develop algorithm of Distance Vector Routing and explain the same.	[10]	3	L3
4	a	Construct a detailed diagram of TCP and UDP headers and show how each field contributes to packet delivery in client-server communication.	[10]	4	L3
5	a	Explain SSH and its components with neat diagram.	[05]	4	L2
5	b	Explain the structure and purpose of HTTP request and response message formats.	[05]	4	L2
6	a	Briefly explain Recursive Resolution & Iterative Resolution in DNS.	[05]	4	L2
6	b	Explain three-way handshaking of TCP connection establishment.	[05]	4	L2

CI CO HoD

1(a)

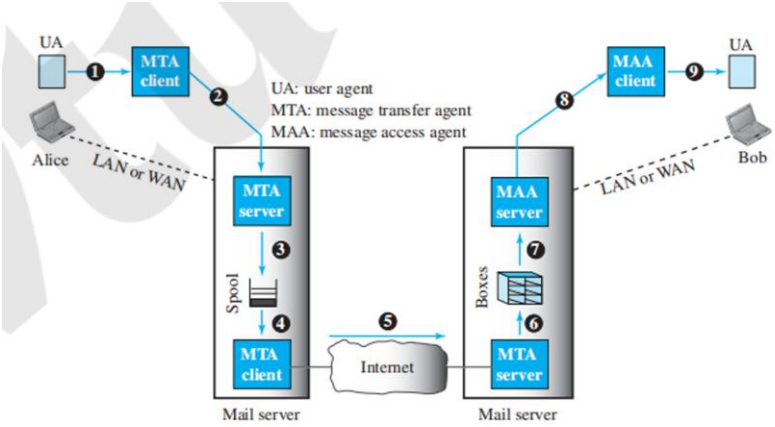
Illustrate how the architecture of electronic mail (email) operates in a real-world communication scenario.

ELECTRONIC MAIL

Electronic mail (or e-mail) allows users to exchange messages. In applications like HTTP or FTP, the server constantly awaits client requests to provide a response, whereas in email, messages are sent and stored until the recipient retrieves them. when Alice sends an email to Bob, a response is optional and, if given, is a separate transaction. Unlike typical client-server setups, Bob doesn't run a server to receive emails, as he might be offline. Instead, intermediate servers handle the client-server functions, allowing users to connect only when they wish to check or send messages.

Architecture

To explain the architecture of email, we present a typical scenario, as illustrated in Figure. An alternative scenario occurs when Alice or Bob is directly connected to their respective mail server, eliminating the need for a LAN or WAN connection; however, this variation does not affect the overall discussion.



In the common scenario, the sender and the receiver of the e-mail, Alice and Bob respectively, are connected via a LAN or a WAN to two mail servers. The administrator has created one mailbox for each user where the received messages are stored. A mailbox is part of a server hard drive, a special file with permission restrictions. Only the owner of the mailbox has access to it. The administrator has also created a queue (spool) to store messages waiting to be sent.

A simple e-mail from Alice to Bob takes nine different steps, as shown in the figure. Alice and Bob use three different agents: a user agent (UA), a message transfer agent (MTA), and a message access agent (MAA).

1. Alice composes an email using her User Agent (UA).

2. The UA sends the message to the Message Transfer Agent (MTA) client on Alice's mail server.

3. The MTA client forwards the message to the MTA server on the same mail server.

4. The email is placed in a spool (queue) on Alice's mail server, awaiting transfer.

5. The MTA client sends the email across the Internet to Bob's mail server.

6. The MTA server on Bob's mail server receives the email.

7. The email is stored in Bob's mailbox on his mail server.

8. Bob's Message Access Agent (MAA) server retrieves the message from the mailbox.

9. Bob uses his MAA client to access the message through his User Agent (UA).

There are two important points we need to emphasize here.

1. Bob cannot directly use the MTA server to receive messages, as this would require him to run the MTA server continuously, keeping his computer and network connection on all the time—an impractical setup, especially with a WAN connection.

2. Unlike the MTA client-server, which "pushes" messages to the server, Bob requires "pull" programs to retrieve messages. This is why he needs Message Access Agent (MAA) programs, allowing him to pull messages from the server when needed.

User Agent

The first component of an electronic mail system is the user agent (UA). A user agent is a software package (program) that composes, reads, replies to, and forwards messages. It also handles local mailboxes on the

user computers. There are two types of user agents: command-driven and GUI-based. A command-driven user agent normally accepts a one-character command from the keyboard to perform its task. For example, a user can type the character r, at the command prompt, to reply to the sender of the message, or type the character R to reply to the sender and all recipients. Some examples of command driven user agents are mail, pine, and elm. Modern user agents are GUI-based. They contain graphical user interface (GUI) components that allow the user to interact with the software by using both the keyboard and the mouse. Some examples of GUI-based user agents are Eudora and Outlook.

Sending Mail

To send mail, the user, through the UA, creates mail that looks very similar to postal mail. It has an envelope and a message. The envelope usually contains the sender address, the receiver address, and other information. The message contains the header and the body. The header of the message defines the sender, the receiver, the subject of the message, and some other information. The body of the message contains the actual information to be read by the recipient.

Receiving Mail

The user agent is triggered by the user (or a timer). If a user has mail, the UA informs the user with a notice. If the user is ready to read the mail, a list is displayed in which each line contains a summary of the information about a particular message in the mailbox. The summary usually includes the sender mail address, the subject, and the time the mail was sent or received. The user can select any of the messages and display its contents on the screen.

Addresses

To deliver mail, a mail handling system must use an addressing system with unique addresses. In the Internet, the address consists of two parts: a local part and a domain name, separated by an @ sign. The local part of an email address specifies the user's mailbox, while the domain part identifies the mail server or exchanger for sending and receiving mail.

Mailing List or Group List

Electronic mail allows one name, an alias, to represent several different e-mail addresses; this is called a mailing list. Every time a message is to be sent, the system checks the recipient’s name against the alias database; if there is a mailing list for the defined alias, separate messages, one for each entry in the list, must be prepared and handed to the MTA.

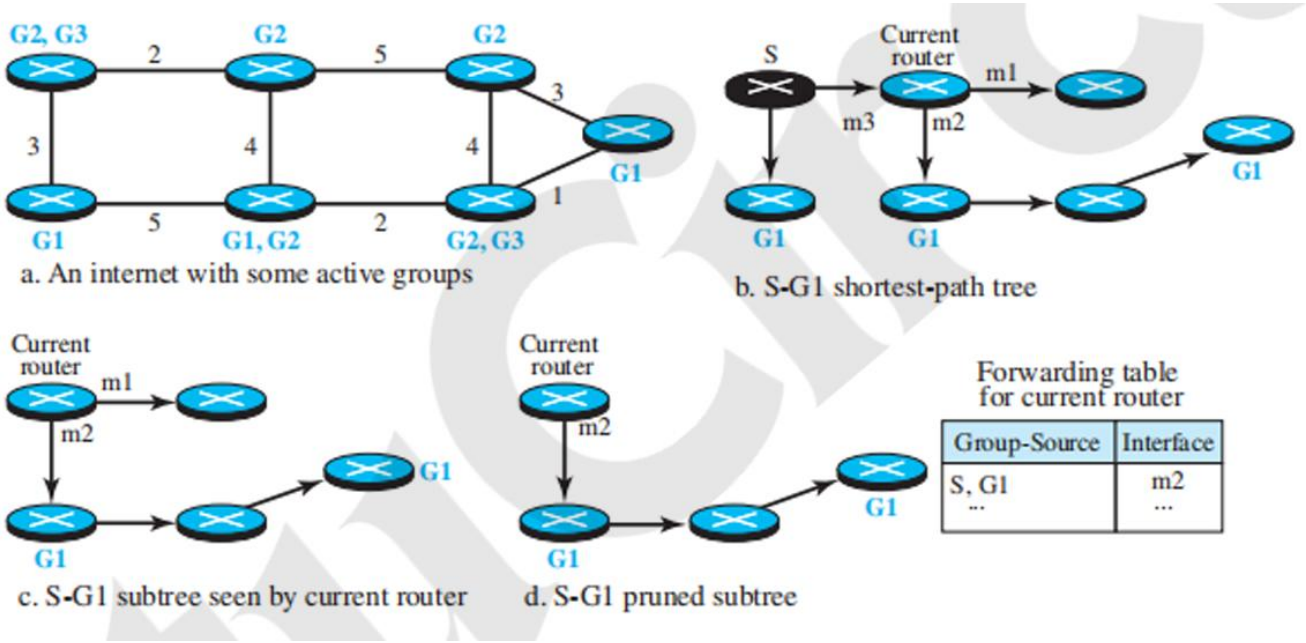
Message Transfer Agent: SMTP

An e-mail is an application that needs three uses of client-server paradigms to accomplish its task. It is important that we distinguish these three when we are dealing with e-mail. Figure shows these three client-server applications. We refer to the first and the second as Message Transfer Agents (MTAs), the third as Message Access Agent (MAA).

The formal protocol that defines the MTA client and server in the Internet is called Simple Mail Transfer Protocol (SMTP).

2(a)

Explain MOSPF with an example and suitable diagram.



Multicast Link State (MOSPF)

Path First (OSPF) protocol, which is used in unicast routing. It also uses the source-based tree approach to multicasting. If the internet is running a unicast link-state routing algorithm, the idea can be extended to provide a multicast link-state routing algorithm.
To extend unicasting to multicasting, each router needs to have another database, as with the case of unicast distance-vector routing, to show which interface has an active member in a particular group. Now a router goes through the following steps to forward a multicast packet received from source S and to be sent to destination G (a group of recipients):

---

1. Create Shortest-Path Tree Using Dijkstra Algorithm

The router uses the Dijkstra algorithm to create a shortest-path tree with S as the root and all destinations in the internet as the leaves. Note that this shortest-path tree is different from the one the router normally uses for unicast forwarding, in which the root of the tree is the router itself.
In this case, the root of the tree is the source of the packet defined in the source address of the packet. The router is capable of creating this tree because it has the LSDB, the whole topology of the internet; the Dijkstra algorithm can be used to create a tree with any root, no matter which router is using it. The point we need to remember is that the shortest-path tree created this way depends on the specific source. For each source we need to create a different tree.

---

2. Locate the Router in the Shortest-Path Tree

The router finds itself in the shortest-path tree created in the first step. In other words, the router creates a shortest-path subtree with itself as the root of the subtree.

---

3. Prune the Broadcast Subtree into a Multicast Tree

The shortest-path subtree is actually a broadcast subtree with the router as the root and all networks as the leaves. The router now uses a strategy similar to the one we describe in the case of DVMRP to prune the broadcast tree and to change it to a multicast tree. The IGMP protocol is used to find the information at the leaf level. MOSPF has added a new type of link state update packet that floods the membership to all routers. The router can use the information it receives in this way and prune the broadcast tree to make the multicast tree.

---

4. Forward the Multicast Packet Through Required Interfaces

The router can now forward the received packet out of only those interfaces that correspond to the branches of the multicast tree. We need to make certain that a copy of the multicast packet reaches all networks that have active members of the group and that it does not reach those networks that do not. Figure shows an example of using the steps to change a graph to a multicast tree. For simplicity, we have not shown the network, but we added the groups to each router. The figure shows how a source-based tree is made with the source as the root and changed to a multicast subtree with the root at the current router.

3(a)

Develop algorithm of Distance Vector Routing and explain the same.

The Distance-Vector (DV) Routing Algorithm

➔ The distance vector (DV) algorithm is iterative, asynchronous, and distributed.

➔ It is iterative that the process continues on until no more information is exchanged between neighbours.

➔ The algorithm is asynchronous in that it does not require all of the nodes to operate in lockstep with each other.

➔ Let dx(y) be the cost of the least-cost path from node x to node y. Then the least costs are related by the celebrated Bellman-Ford equation, namely,

dx(y) = minv { c(x,v) + dv(y) }

➔ With the DV algorithm, each node x maintains the following routing information:

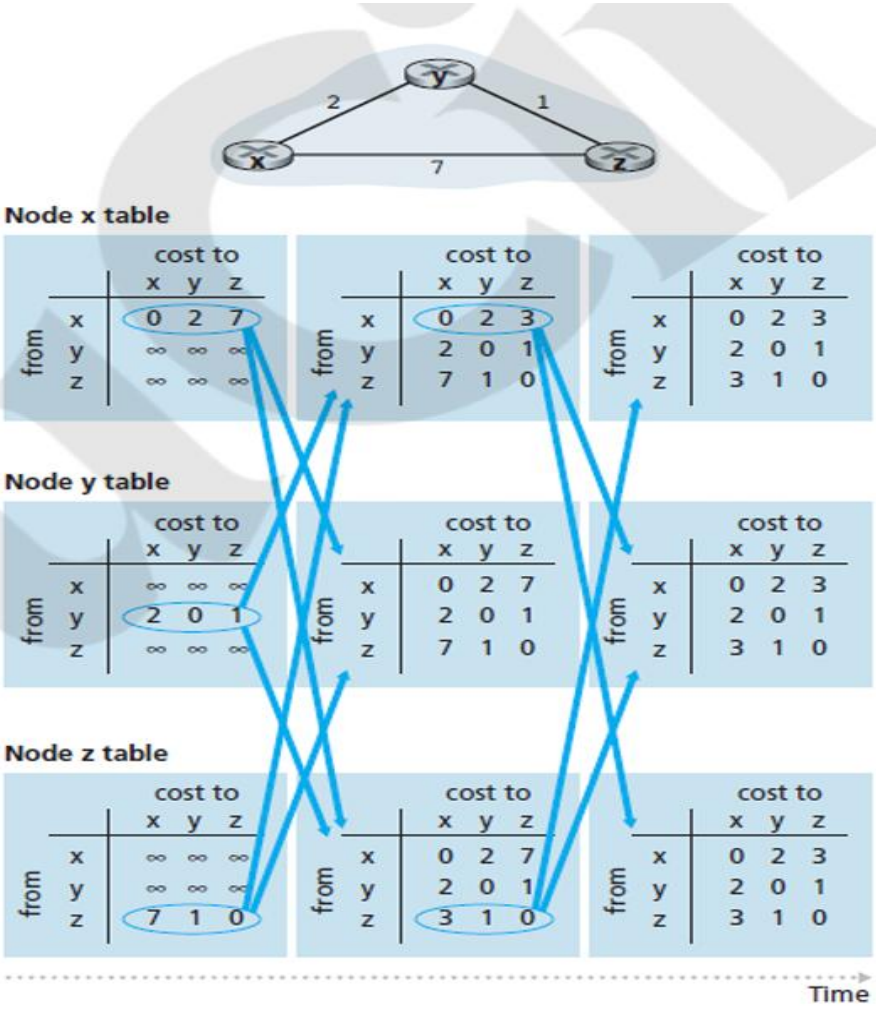
o For each neighbour v, the cost c(x,v) from x to directly attached neighbour, v

o Node x's distance vector, that is, Dx = [Dx(y): y in N], containing x's estimate of its cost to all destinations, y, in N

o The distance vectors of each of its neighbours, that is, Dv = [Dv(y): y in N] for each neighbour v of x

```
1  Initialization:
2     for all destinations y in N:
3        Dx(y) = c(x,y)    /* if y is not a neighbor then c(x,y) = ∞ */
4     for each neighbor w
5        Dw(y) = ? for all destinations y in N
6     for each neighbor w
7        send distance vector Dx = [Dx(y): y in N] to w
8
9  loop
10    wait (until I see a link cost change to some neighbor w or
11          until I receive a distance vector from some neighbor w)
12
13    for each y in N:
14       Dx(y) = minv{c(x,v) + Dv(y)}
15
16    if Dx(y) changed for any destination y
17       send distance vector Dx = [Dx(y): y in N] to all neighbors
18
19 forever
```

Figure illustrates the operation of the DV algorithm for the simple three node network shown at the top of the figure.



Distance-vector (DV) algorithm

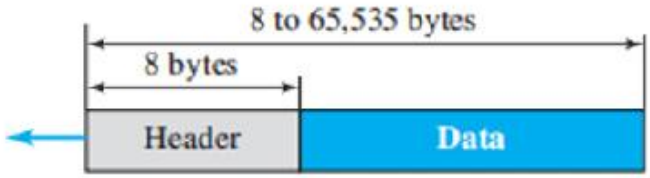
4(a)

Construct a detailed diagram of TCP and UDP headers and show how each field contributes to packet delivery in client-server communication.

The transport layer is situated between the application layer and the network layer, providing **process-to-process communication** between application layers at the local and remote hosts. The delivery of messages to the correct process (rather than just the destination computer, which is the network layer's responsibility) is handled by transport-layer protocols like TCP and UDP, using **port numbers**.

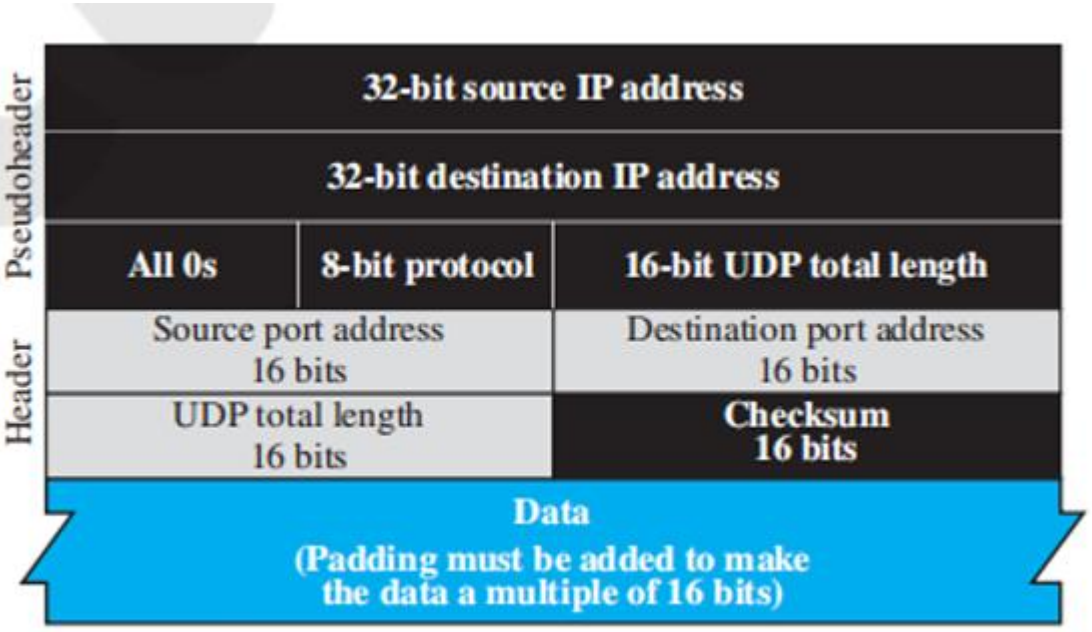
In client-server communication, a combination of an IP address and a port number, called a **socket address**, is required at both ends to define the client and server processes uniquely and make a connection. The transport layer handles encapsulation (adding a header on the sender's side) and decapsulation (removing the header and delivering the message on the receiver's side).

Below are detailed descriptions of the UDP and TCP headers and how their fields contribute to this delivery mechanism.



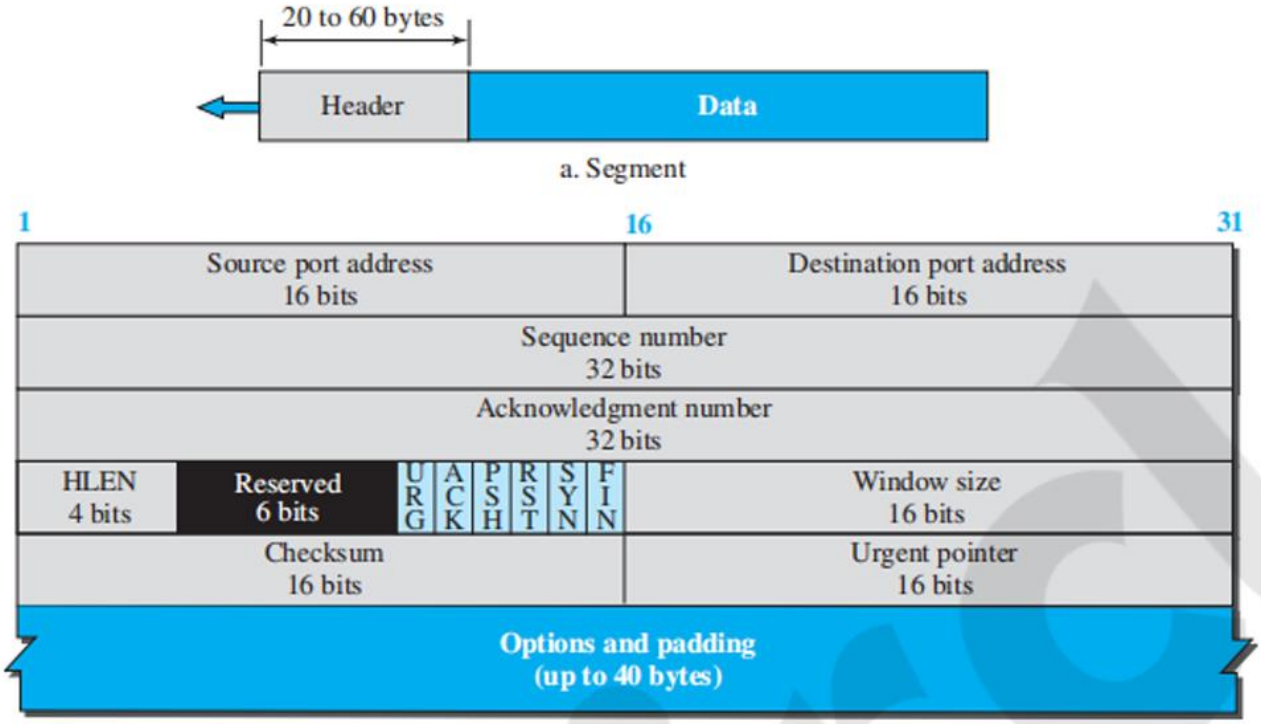
a. UDP user datagram

b. Header format



User Datagram Protocol (UDP) Header

UDP is a **connectionless, unreliable transport protocol** that uses a minimum of overhead. Its packets, known as **user datagrams**, have a fixed-size header of **8 bytes**, consisting of four 16-bit fields.

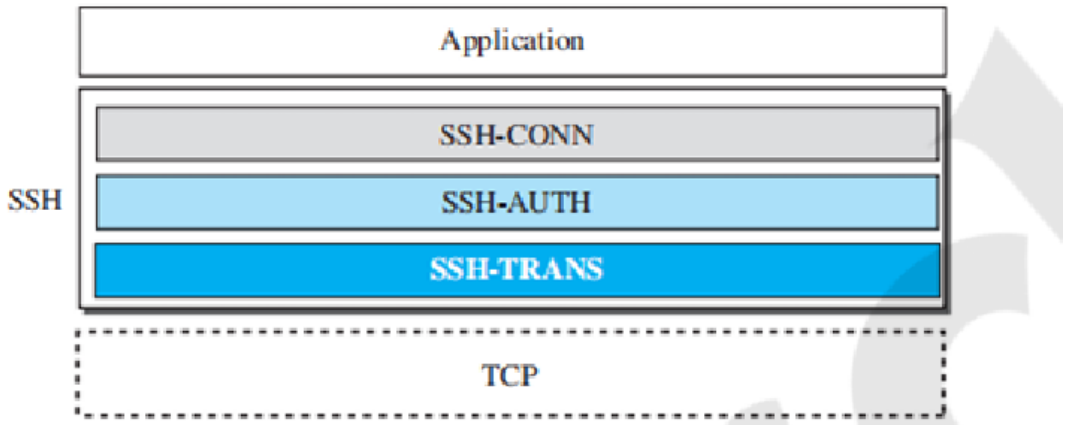


Transmission Control Protocol (TCP) Header

TCP is a **reliable connection-oriented protocol**. TCP packets, called **segments**, use a combination of sequence numbers, acknowledgments, and control flags to manage reliability, flow, and connection state. The TCP header length is variable, ranging from a minimum of **20 bytes** (if no options are present) up to **60 bytes** (if options are included)

5(a)

Explain SSH and its components with neat diagram.



SECURE SHELL (SSH)

Although Secure Shell (SSH) is a secure application program that can be used today for several purposes such as remote logging and file transfer, it was originally designed to replace TELNET. There are two

versions of SSH: SSH-1 and SSH-2, which are totally incompatible. The first version, SSH-1, is now deprecated because of security flaws in it.

Components

SSH is an application-layer protocol with three components.

SSH Transport-Layer Protocol (SSH-TRANS)

Since TCP is not a secure transport-layer protocol, SSH uses an additional protocol called SSH-TRANS to create a secure channel on top of TCP. The process begins with the client and server establishing an insecure connection via TCP. They then exchange security parameters to establish a secure channel using SSH-TRANS.

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, which improves the efficiency of the system and makes attack more difficult.

SSH Authentication Protocol (SSH-AUTH)

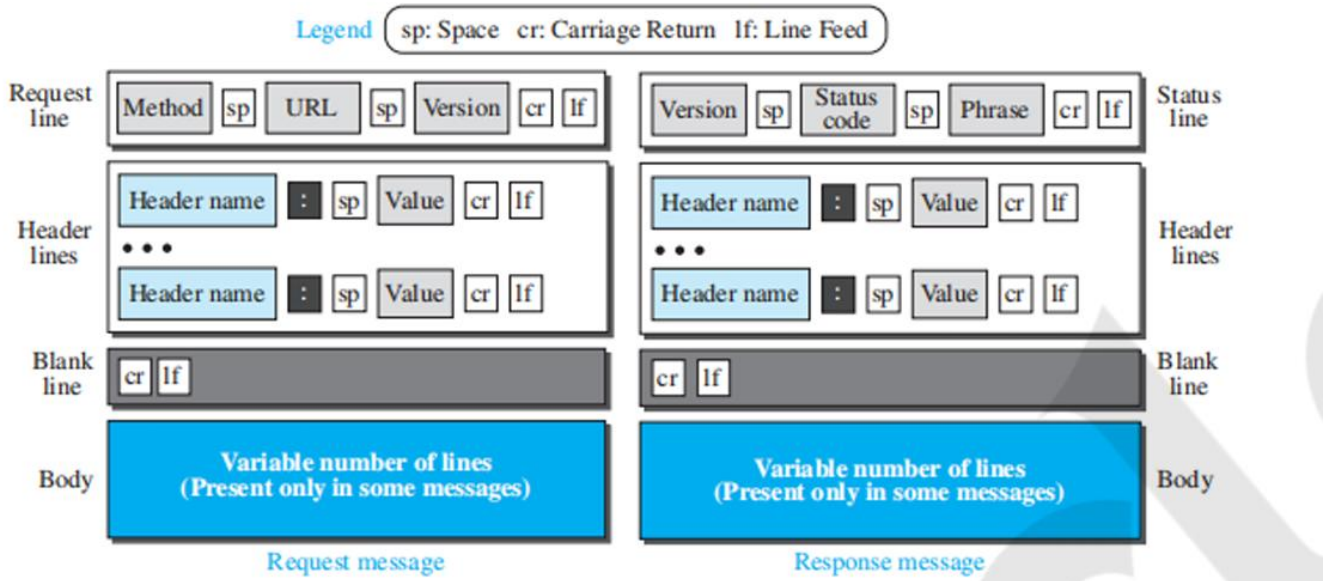
After establishing a secure channel and authenticating the server, SSH proceeds to authenticate the client. The client sends a request to the server with the username, server name, authentication method, and required data. The server then responds with either a success message, confirming authentication, or a failure message, prompting the client to try again with a new request. This process is similar to client authentication in SSL.

SSH Connection Protocol (SSH-CONN)

Once the secure channel is established and both the server and client are authenticated, SSH uses the SSH-CONN protocol. One of its key features is multiplexing, which allows the client to create multiple logical channels over the secure channel. Each channel can be used for different purposes, such as remote logging or file transfer.

5(b)

Explain the structure and purpose of HTTP request and response message formats.



Message Formats

The HTTP protocol defines the format of the request and response messages.

Request Message

Method:

There are five HTTP methods:

- **GET:** The GET method is used when the browser requests an object, with the requested object identified in the URL field.
- **POST:** With a POST message, the user is still requesting a Web page from the server, but the specific contents of the Web page depend on what the user entered into the form fields. If the value of the method field is POST, then the entity body contains what the user entered into the form fields.
- **PUT:** The PUT method is also used by applications that need to upload objects to Web servers.
- **HEAD:** Used to retrieve header information. It is used for debugging purpose.
- **DELETE:** The DELETE method allows a user, or an application, to delete an object on a Web server.

URL:

Specifies URL of the requested object

Version:

This field represents HTTP version, usually HTTP/1.1

Header line:

Ex:
Host: www.someschool.edu
Connection: close

User-agent: Mozilla/5.0
Accept-language: fr

The header line **Host:www.someschool.edu** specifies the host on which the object resides.
By including the **Connection:close** header line, the browser is telling the server that it doesn't want to bother with persistent connections; it wants the server to close the connection after sending the requested object.
The **User-agent:** header line specifies the user agent, that is, the browser type that is making the request to the server. Here the user agent is Mozilla/5.0, a Firefox browser.
The **Accept-language:** header indicates that the user prefers to receive a French version of the object, if such an object exists on the server; otherwise, the server should send its default version.

Response Message

Example

HTTP/1.1 200 OK
Connection: close
Date: Tue, 09 Aug 2011 15:44:04 GMT
Server: Apache/2.2.3 (CentOS)
Content-Length: 6821
Last-Modified: Tue, 09 Aug 2011 15:11:03 GMT
Content-Type: text/html
(data data data data data ...)

Status Line

The status line has three fields: the protocol version field, a status code, and a corresponding status message.

Version is HTTP/1.1
The status code and associated phrase indicate the result of the request. Some common status codes and associated phrases include:

1. **200 OK:** Request succeeded and the information is returned in the response.
2. **301 Moved Permanently:** Requested object has been permanently moved; the new URL is specified in Location: header of the response message. The client software will automatically retrieve the new URL.
3. **400 Bad Request:** This is a generic error code indicating that the request could not be understood by the server.
4. **404 Not Found:** The requested document does not exist on this server.
5. **505 HTTP Version Not Supported:** The requested HTTP protocol version is not supported by the server.

Header fields:

➢ The server uses the **Connection: close** header line to tell the client that it is going to close the TCP connection after sending the message.

➢ The **Date:** header line indicates the time and date when the HTTP response was created and sent by the server.

➢ The **Server:** header line indicates that the message was generated by an Apache Web server; it is analogous to the User-agent: header line in the HTTP request message.

➢ The **Last-Modified:** header line indicates the time and date when the object was created or last modified.

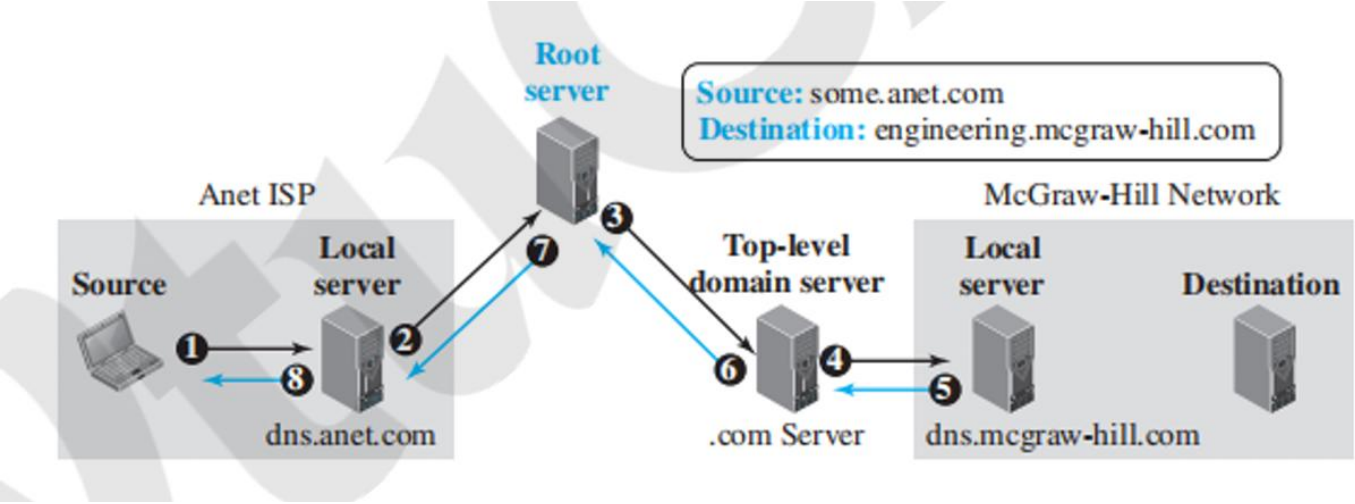
➢ The **Content-Length:** header line indicates the number of bytes in the object being sent.

➢ The **Content-Type:** header line indicates that the object in the entity body is HTML text.

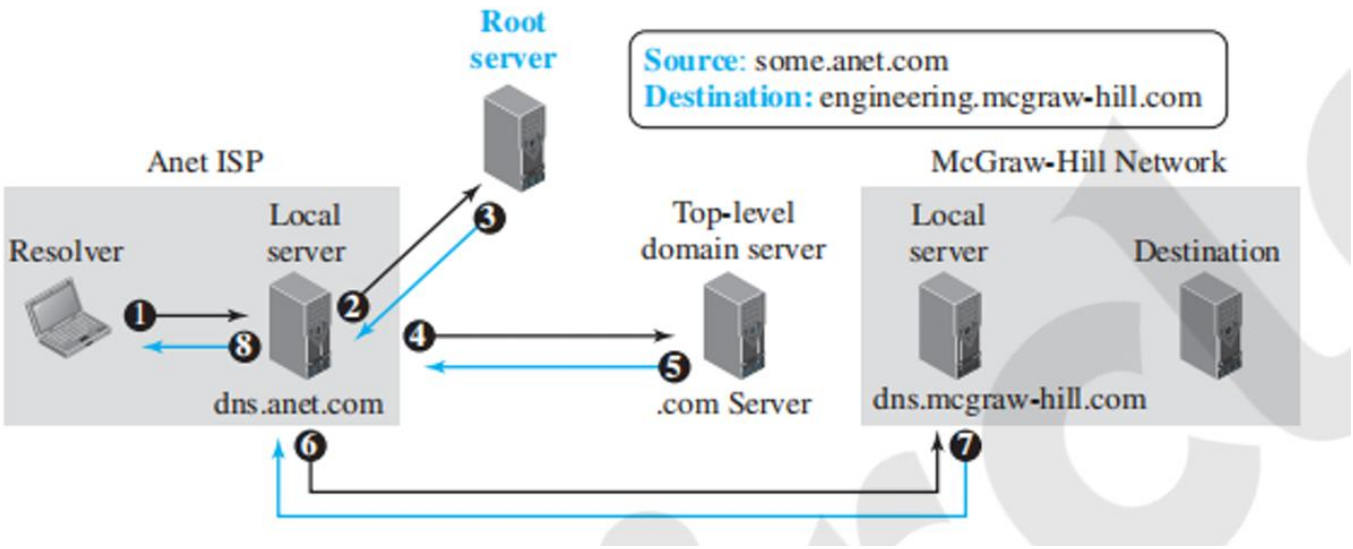
6(a)

Briefly explain Recursive Resolution & Iterative Resolution in DNS.

Recursive Resolution In a recursive DNS resolution, an application on a host (e.g., some.anet.com) needs the IP address of another host (e.g., engineering.mcgraw-hill.com) to send a message. The source host's DNS resolver (client) queries its local DNS server (dns.anet.com), which doesn't have the address and forwards the request to a root DNS server. The root server, lacking the specific mapping, knows the relevant top-level domain server (e.g., for .com) and forwards the query there. This server, not having the exact address, directs the query to McGraw-Hill's local DNS server (dns.mcgraw-hill.com), which finally provides the destination IP. This IP address is sent back step-by-step—from McGraw-Hill's DNS server to the top-level DNS server, then to the root server, the ISP's DNS server (which caches it), and ultimately back to the source host.

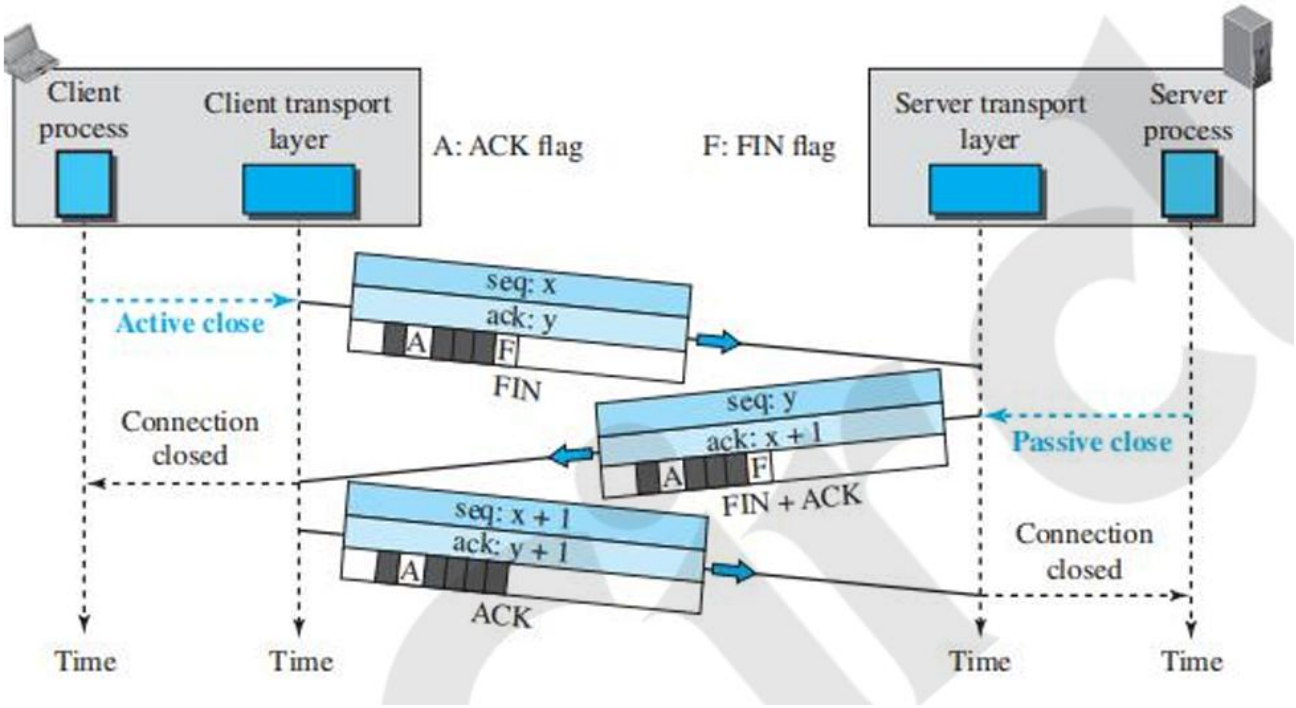


Iterative Resolution In iterative resolution, each server that does not know the mapping sends the IP address of the next server back to the one that requested it. Figure shows the flow of information in an iterative resolution. Normally the iterative resolution takes place between two local servers; the original resolver gets the final answer from the local server. Note that the messages shown by events 2, 4, and 6 contain the same query. However, the message shown by event 3 contains the IP address of the top-level domain server, the message shown by event 5 contains the IP address of the McGraw-Hill local DNS server, and the message shown by event 7 contains the IP address of the destination. When the Anet local DNS server receives the IP address of the destination, it sends it to the resolver (event 8).



6(b)

Explain three-way handshaking of TCP connection establishment.



Three-Way Handshaking

The connection establishment in TCP is called three-way handshaking. For example, an application program, called the client, wants to make a connection with another application program, called the server, using TCP as the transport-layer protocol.

The process starts with the server. The server program tells its TCP that it is ready to accept a connection. This request is called a passive open. Although the server TCP is ready to accept a connection from any machine in the world, it cannot make the connection itself.

The client program issues a request for an active open. A client that wishes to connect to an open server tells its TCP to connect to a particular server. TCP can now start the three-way handshaking process, as shown in Figure.

The three steps in this phase are as follows:

1. **Client → SYN**
   The client sends the first segment, a SYN segment, in which only the SYN flag is set. This segment is for synchronization of sequence numbers. This sequence number is called the initial sequence number (ISN). This segment does not contain an acknowledgment number. The SYN segment is a control segment and carries no data.

2. **Server → SYN + ACK**
   The server sends the second segment, a SYN + ACK segment with two flag bits set as: SYN and ACK. This segment has a dual purpose.

   - It is a SYN segment for communication in the other direction.
   - It acknowledges the receipt of the SYN segment from the client by setting the ACK flag and displaying the next sequence number it expects to receive from the client.
     Because the segment contains an acknowledgment, it also needs to define the receive.

3. **Client → ACK**
   The client sends the third segment. This is just an ACK segment. It acknowledges the receipt of the second segment with the ACK flag and acknowledgment number field.