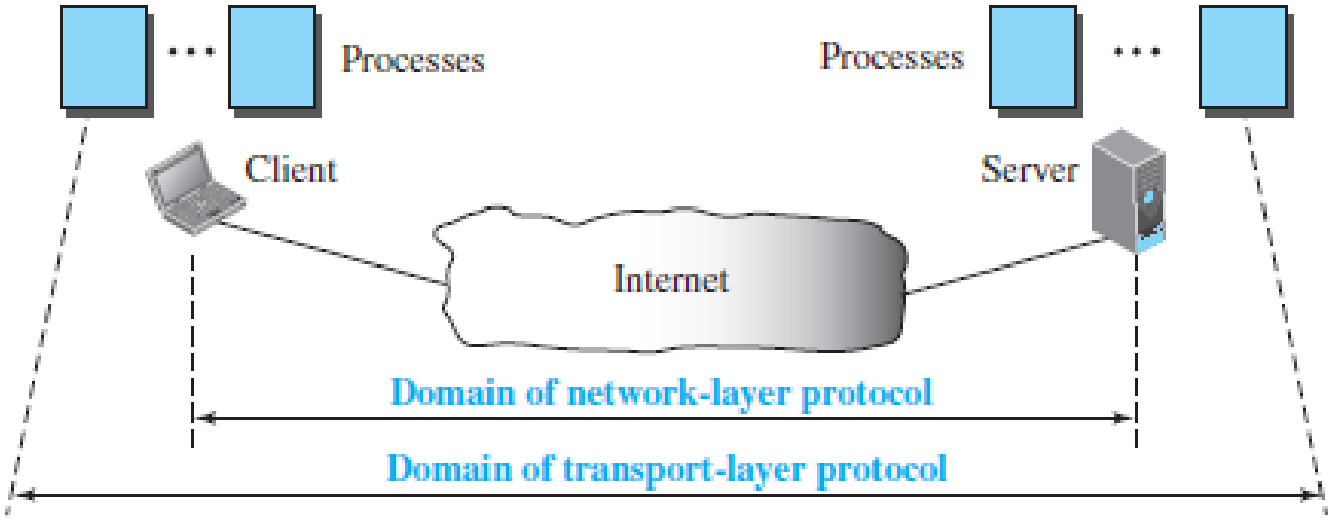
Transport-Layer Services

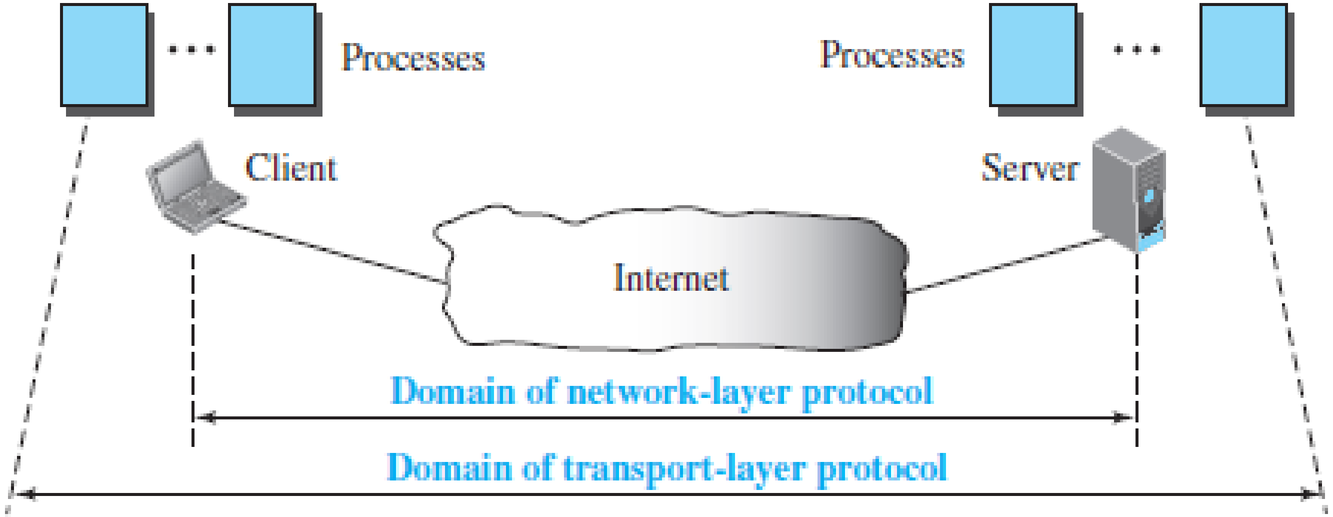
- ***Process-to-Process Communication:*** A process is an application-layer entity (running program) that uses the services of the transport layer.

- The network layer is responsible for communication at the computer level (host-to-host communication). A network-layer protocol can deliver the message only to the destination computer.


Processes
Processes
Client
Server
Internet
Domain of network-layer protocol
Domain of transport-layer protocol

- However, this is an incomplete delivery, as the message still needs to be handed to the correct process.

- A transport-layer protocol is responsible for delivery of the message to the appropriate process. TL provides end to end or process to process communication


Processes
Processes
Client
Server
Internet
Domain of network-layer protocol
Domain of transport-layer protocol

Q Which of the following functionalities must be implemented by a transport protocol over and above the network protocol? **(Gate-2003) (1 Marks)**

(A) Recovery from packet losses

(B) Detection of duplicate packets

(C) Packet delivery in the correct order

(D) End to end connectivity

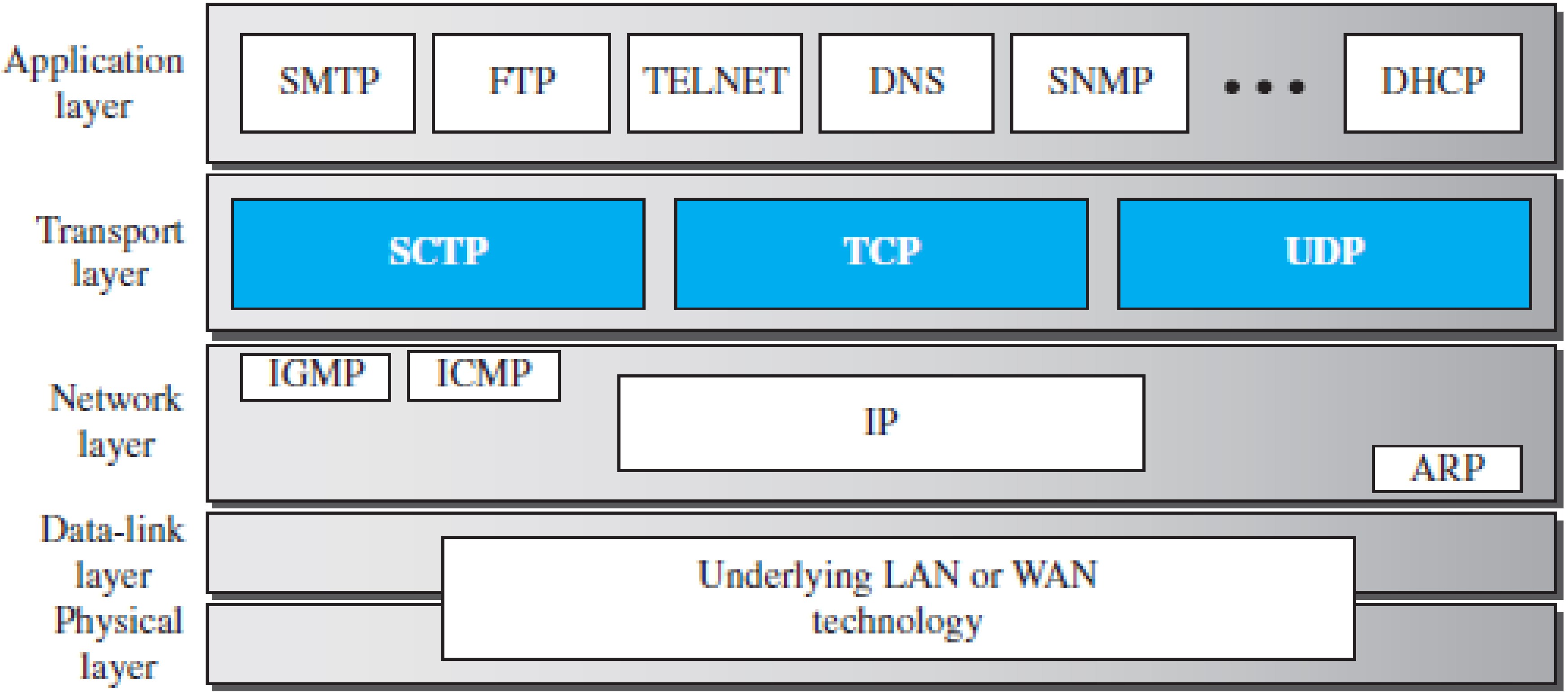
Break

- A transport layer protocol can be either connectionless or connection-oriented.
    - A connectionless transport layer treats each segment as an independent packet and delivers it to the transport layer at the destination machine.
    - A connection-oriented transport layer makes a connection with the transport layer at the destination machine first before delivering the packets. After all the data is transferred, the connection is terminated.
- The transport layer may be responsible for flow and error control. However, flow and error control at this layer is performed end to end rather than across a single link.

- Reliable Versus Unreliable
  - If the application layer program needs reliability, we use a reliable transport layer protocol by implementing flow and error control at the transport layer. This means a slower and more complex service.
  - On the other hand, if the application program does not need reliability because it uses its own flow and error control mechanism or it needs fast service or the nature of the service does not demand flow and error control (real-time applications), then an unreliable protocol can be used.

- In the Internet, there are three common different transport layer protocols.
  - UDP is connectionless and unreliable;
  - TCP and SCTP are connection oriented and reliable. These three can respond to the demands of the application layer programs.

- TCP offers *full-duplex service,* where data can flow in both directions at the same time.



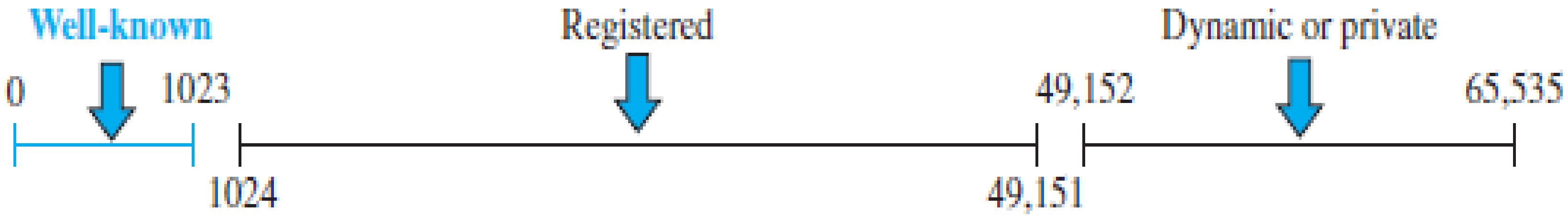
Break

Addressing: Port Numbers

- For communication, we must define the local host, local process, remote host, and remote process.
- Local and Remote host are defined by IP Addresses. To define the processes inside a host, we need second identifiers, called port numbers, they are 16-bits integers ranging from (0 to $2^{16} – 1$) or (0 to 65535).

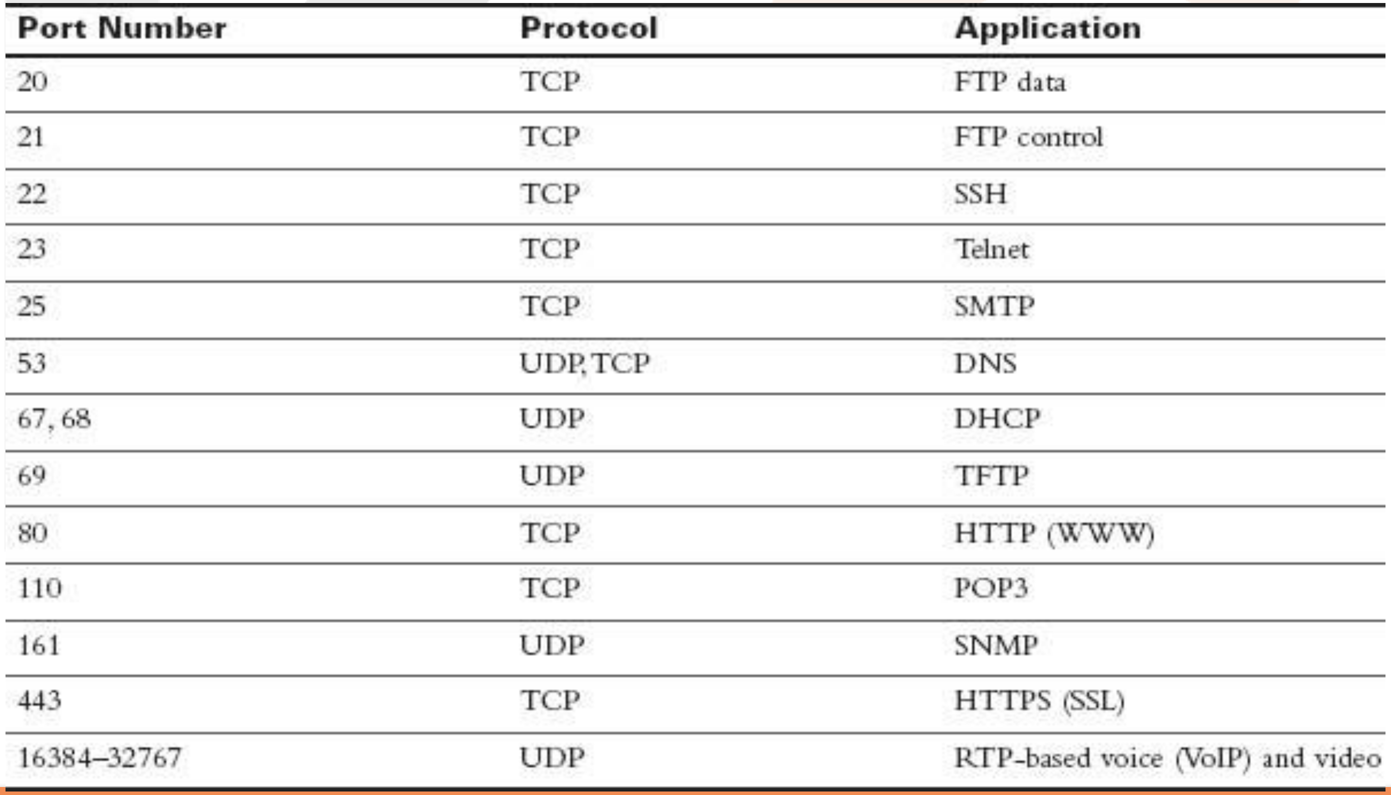
ICANN Ranges

- ICANN (Internet Corporation for Assigned Names and Numbers) has divided the port numbers into three ranges: well-known, registered, and dynamic (or private). Till 2001



- The IANA (Internet Assigned Number Authority) has divided the port numbers into three ranges: well known, registered, and dynamic (or private).

- **Well-known ports:**
  - The ports ranging from 0 to 1023 are assigned and controlled by IANA. These are the well-known ports.
  - The server process must also define itself with a port number. This port number, however, cannot be chosen randomly as the client has to request the data from server.
  - TCP/IP has decided to use universal port numbers for servers; these are called ***well-known port numbers.***



Port Number	Protocol	Application
20	TCP	FTP data
21	TCP	FTP control
22	TCP	SSH
23	TCP	Telnet
25	TCP	SMTP
53	UDP, TCP	DNS
67, 68	UDP	DHCP
69	UDP	TFTP
80	TCP	HTTP (WWW)
110	TCP	POP3
161	UDP	SNMP
443	TCP	HTTPS (SSL)
16384–32767	UDP	RTP-based voice (VoIP) and video

- **Registered ports:**
  - The ports ranging from 1024 to 49,151 are not assigned or controlled by IANA. They can only be registered with IANA to prevent duplication.

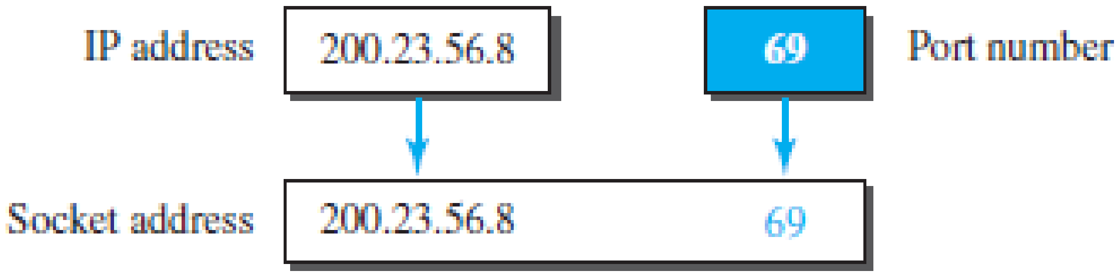
- **Dynamic/Ephemeral ports:**
  - The ports ranging from 49,152 to 65,535 are neither controlled nor registered. They can be used by any process.
  - The client program defines itself with a port number, chosen randomly by the transport layer software running on the client host.
  - Ephemeral means “short-lived” and is used because the life of a client is normally short.

Break

Socket Addresses

- A transport-layer protocol in the TCP suite needs both the IP address and the port number, at each end, to make a connection. To use the services of the transport layer in the Internet, we need a pair of socket addresses: the client socket address and the server socket address.

- The combination of an IP address and a port number is called a ***socket address.***

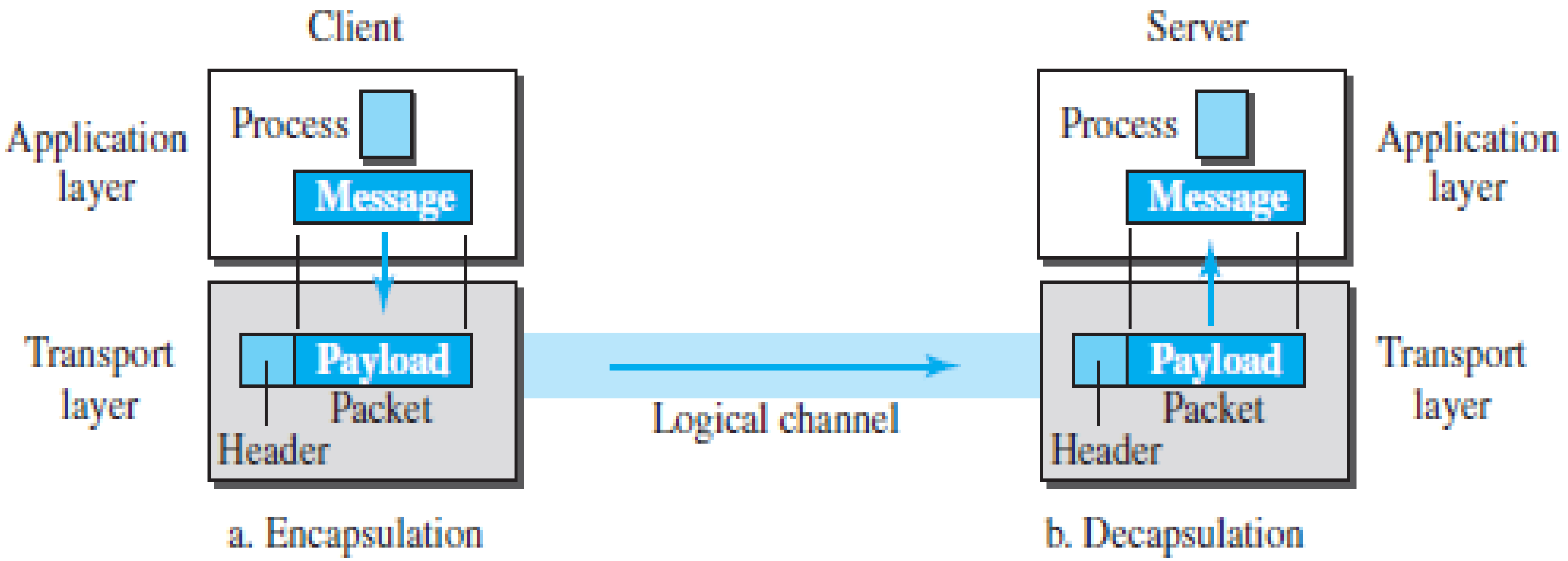


Encapsulation and Decapsulation

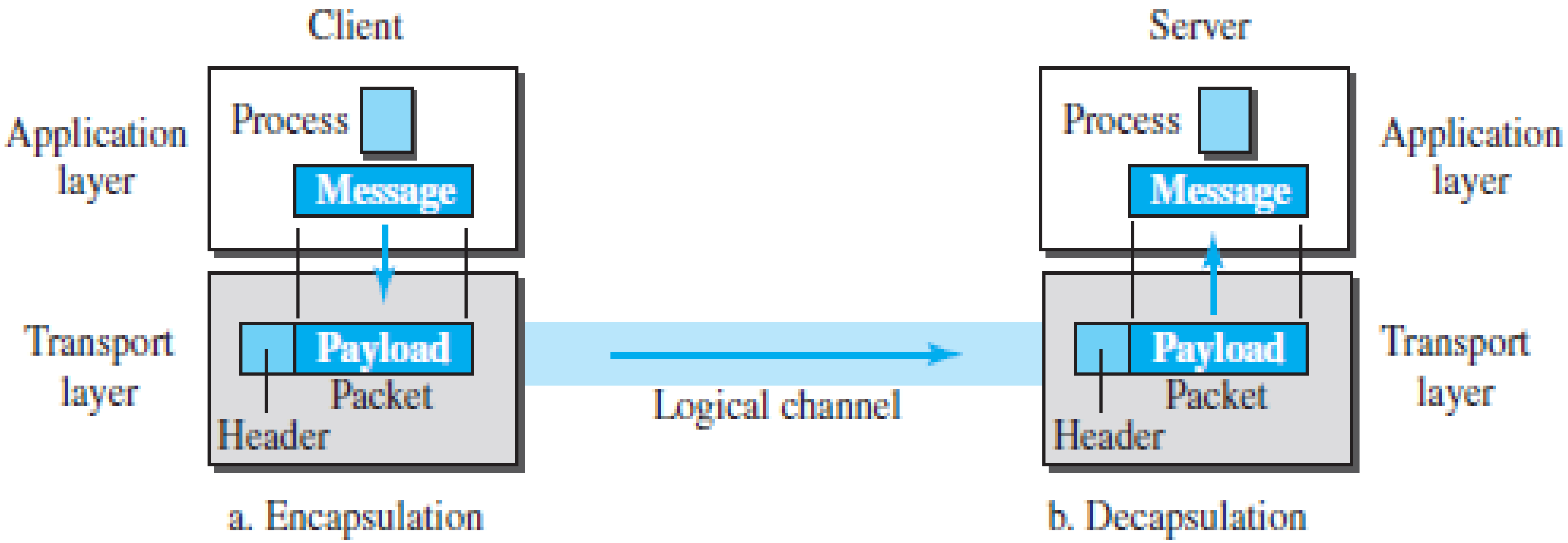
- To send a message from one process to another, the transport-layer protocol encapsulates and decapsulates messages.

- Encapsulation happens at the sender site. When a process has a message to send, it passes the message to the transport layer along with a pair of socket addresses.

- The transport layer receives the data and adds the transport-layer header. The packets at the transport layer in the Internet are called ***segments.***



- Decapsulation happens at the receiver site.
- When the message arrives at the destination transport layer, the header is dropped and the transport layer delivers the message to the process running at the application layer.



a. Encapsulation

b. Decapsulation

Q What is the maximum size of data that the application layer can pass on to the TCP layer below? **(Gate-2008) (1 Marks)**

(A) Any size

(B) 2^{16} bytes – size of TCP header

(C) 2^{16} bytes

(D) 1500 bytes

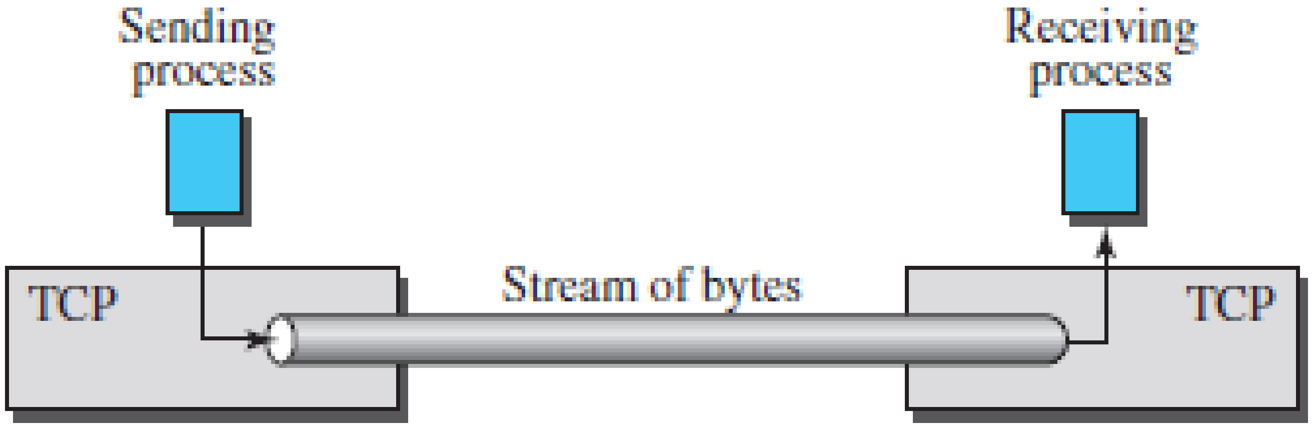
Q A TCP message consisting of 2100 bytes is passed to IP for delivery across two networks. The first network can carry a maximum payload of 1200 bytes per frame and the second network can carry a maximum payload of 400 bytes per frame, excluding network overhead. Assume that IP overhead per packet is 20 bytes. What is the total IP overhead in the second network for this transmission? **(Gate-2004) (2 Marks)**

(A) 40 bytes **(B)** 80 bytes **(C)** 120 bytes **(D)** 160 bytes

Break

TCP (Transmission Control Protocol)

- TCP is a ***reliable connection-oriented protocol***, it must be used in any application where ***reliability is important.***
- It creates a virtual connection between two TCPs to send data. In addition, TCP uses flow and error control mechanisms at the transport level.
- TCP allows the sending process to deliver data as a stream of bytes and allows the receiving process to obtain data as a stream of bytes.
- TCP creates an environment in which the two processes seem to be connected by an imaginary "tube" that carries their data across the Internet.
- It adds connection-oriented and reliability features to the services of IP.

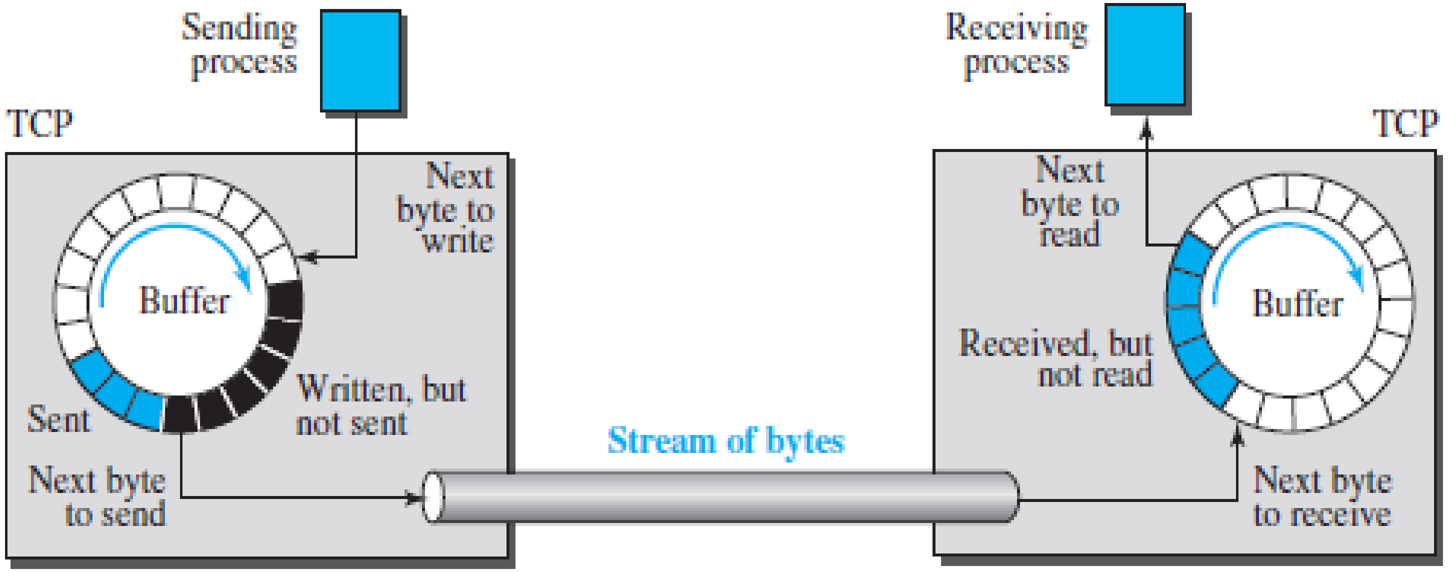


Q In TCP, a unique sequence number is assigned to each **(Gate-2004) (1 Marks)**

(A) byte **(B)** word **(C)** segment **(D)** message

- Connection oriented means some resources will be reserved at the receiver end, like Bandwidth, CPU time, Buffer etc.

- Sending and Receiving Buffers Because the sending and the receiving processes may not write or read data at the same speed, TCP needs buffers for storage.



- **Flow Control**
  - The receiver of the data controls the amount of data that are to be sent by the sender. This is done to prevent the receiver from being overwhelmed with data. The numbering system allows TCP to use a byte-oriented flow control.

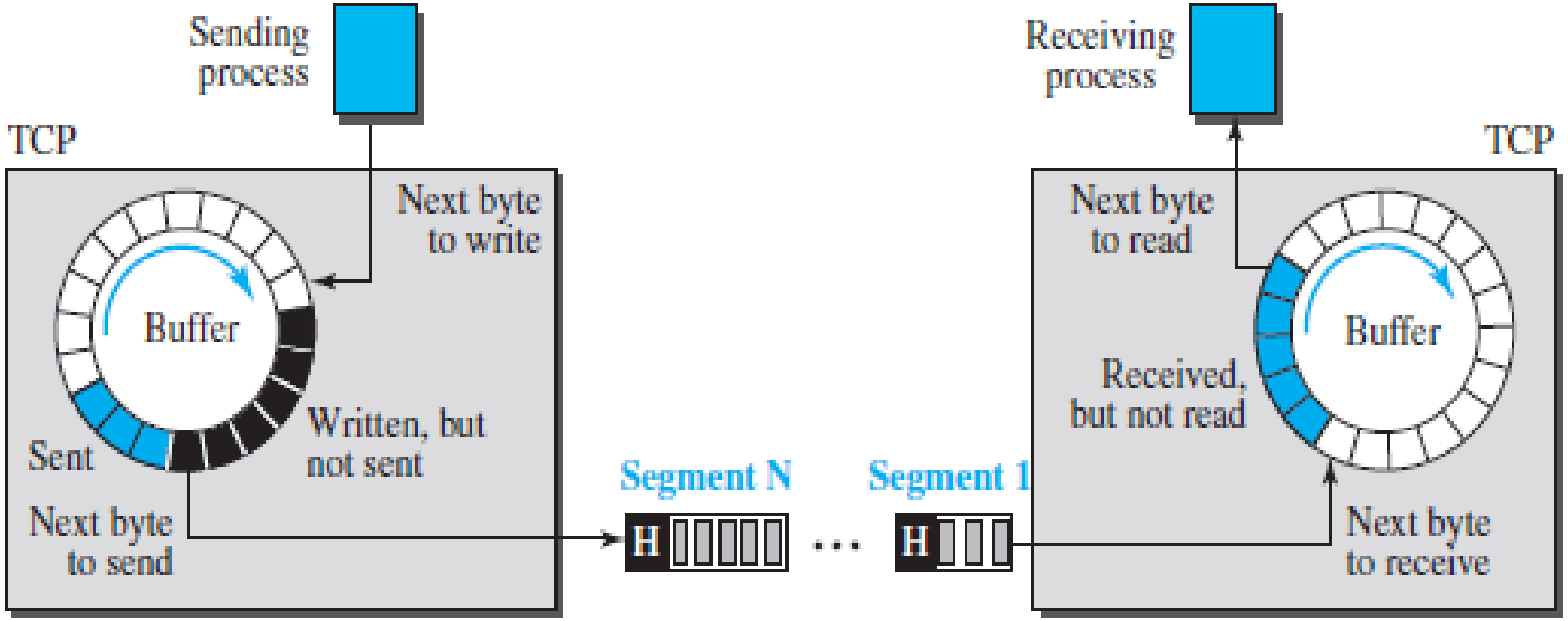
- **Error Control**
  - To provide reliable service, TCP implements an error control mechanism. Although error control considers a segment as the unit of data for error detection (loss or corrupted segments), error control is byte-oriented.

- **Congestion Control**
  - TCP, unlike UDP, takes into account congestion in the network. The amount of data sent by a sender is not only controlled by the receiver (flow control), but is also determined by the level of congestion in the network.

Segments

- The network layer, as a service provider for TCP, needs to send data in packets, not as a stream of bytes. At the transport layer, TCP groups a number of bytes together into a packet called a *segment*.

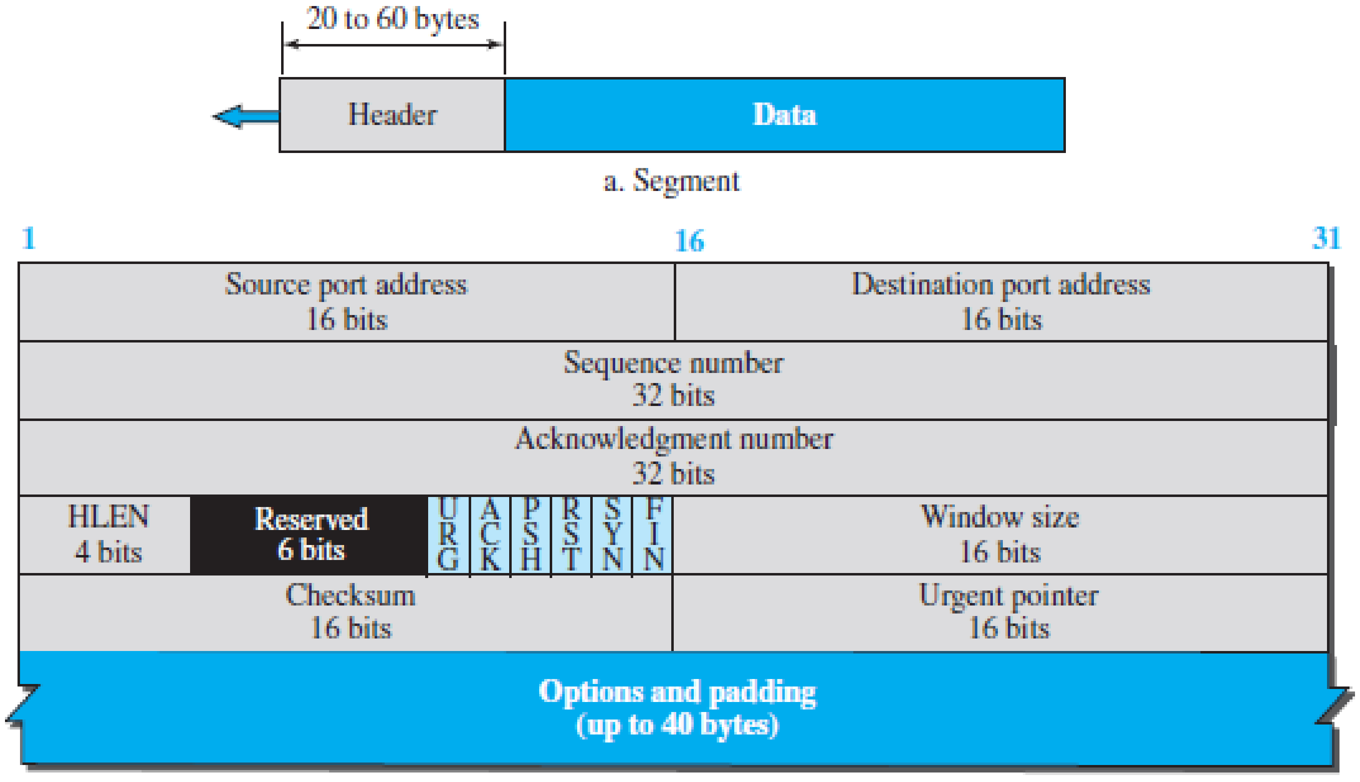
- TCP adds a header to each segment (for control purposes) and delivers the segment to the network layer for transmission. The segments are encapsulated in an IP datagram and transmitted.



Break

TCP Header

- The segment consists of a 20- to 60-byte header, followed by data from the application program. The header is 20 bytes if there are no options and up to 60 bytes if it contains options.



20 to 60 bytes

Header

Data

a. Segment

1

16

31

Source port address
16 bits

Destination port address
16 bits

Sequence number
32 bits

Acknowledgment number
32 bits

HLEN
4 bits

Reserved
6 bits

URG

ACK

PSH

RST

SYN

FIN

Window size
16 bits

Checksum
16 bits

Urgent pointer
16 bits

Options and padding
(up to 40 bytes)

b. Header

Q In the TCP/IP protocol suite, which one of the following is NOT part of the IP header? **(Gate-2004) (2 Marks)**

(A) Fragment Offset

(B) Source IP address

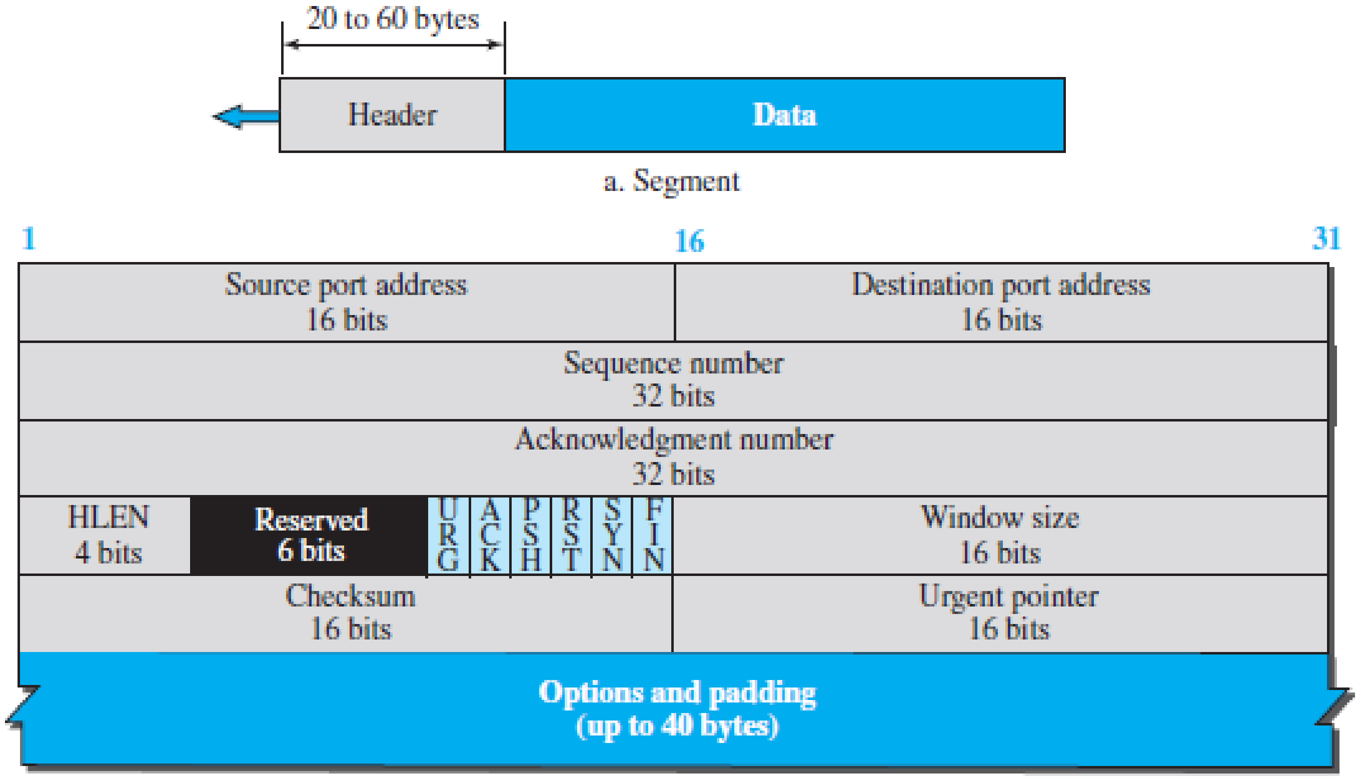
(C) Destination IP address

(D) Destination port number

Source & Destination port address

- **Source port address**. This is a 16-bit field that defines the port number of the application program in the host that is sending the segment.

- **Destination port address**. This is a 16-bit field that defines the port number of the application program in the host that is receiving the segment.



a. Segment

b. Header

Break

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.
- TCP chooses an arbitrary number between 0 and $2^{32} - 1$ for the number of the first byte.

Sequence number

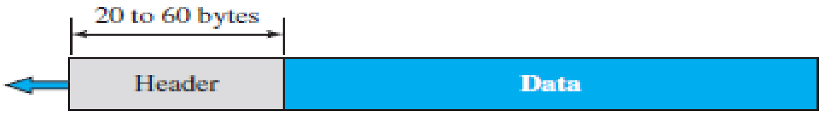
- TCP is a stream transport protocol. To ensure connectivity, each byte to be transmitted is numbered. Sequence number is 32-bit field defines the number assigned to the first byte of data contained in this segment.

- So, maximum number of possible sequence numbers = 2^{32}. These sequence numbers lie in the range [0, $2^{32} – 1$].

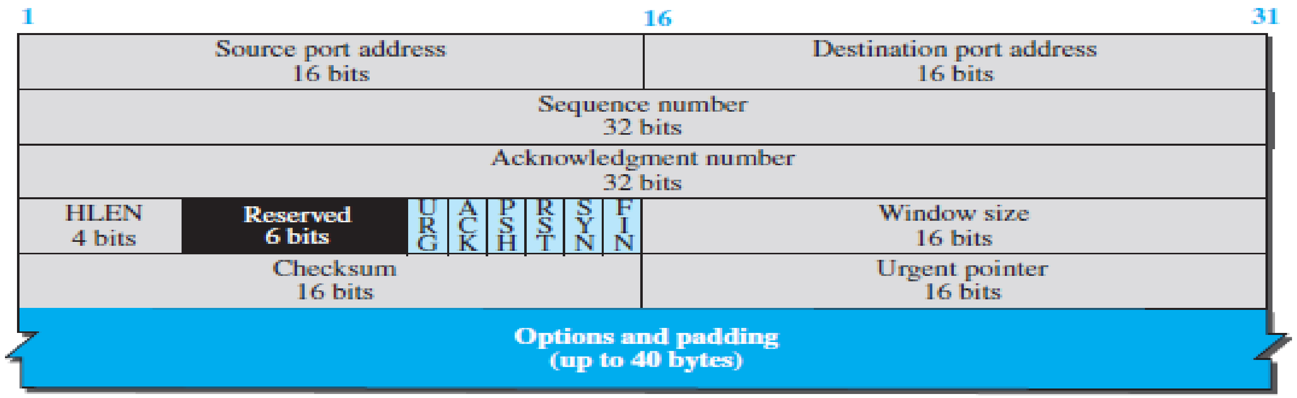
- In IP every packet is counted not Byte, in DLL every bit is counted with HDLC protocol.

- During connection establishment, each party uses a random number generator to create an initial sequence number (ISN), which is usually different in each direction. Sequence number should be started at random, to remove duplication problem.

- The sequence number of any other segment is the sequence number of the previous segment plus the number of bytes (real or imaginary) carried by the previous segment.



a. Segment



b. Header

Example: Suppose a TCP connection is transferring a file of 5000 bytes. The first byte is numbered 10001. What are the sequence numbers for each segment if data are sent in five segments, each carrying 1000 bytes?

- This does not imply that only 2^{32} bytes = 4 GB data can be sent using TCP. The concept of wrap around allows to send unlimited data using TCP.

- After all the 2^{32} sequence numbers are used up and more data is to be sent, the sequence numbers can be wrapped around and used again from the starting.

Wrap Around Time

- Time taken to use up all the 2^{32} sequence numbers is called as **wrap around time**.
- It depends on the bandwidth of the network i.e. the rate at which the bytes go out.
- ***Wrap Around Time $\propto$ 1 / Bandwidth***
  - If bandwidth of the network = x bytes/sec, then
- ***Wrap Around Time = 2^{32}/ x sec.***

Life Time of TCP Segment

- Life time of a TCP segment is 180 seconds or 3 minutes.
- It means after sending a TCP segment, it might reach the receiver taking 3 minutes in the worst case.

- In the last we will do rap around, wrap around time is the time taken to wrap around
  - if WAT > LT then there is no problem
  - if wat < LT then destination will get same sequence no again and again, to solve this problem additional bits can be put in options, called time-stamp.

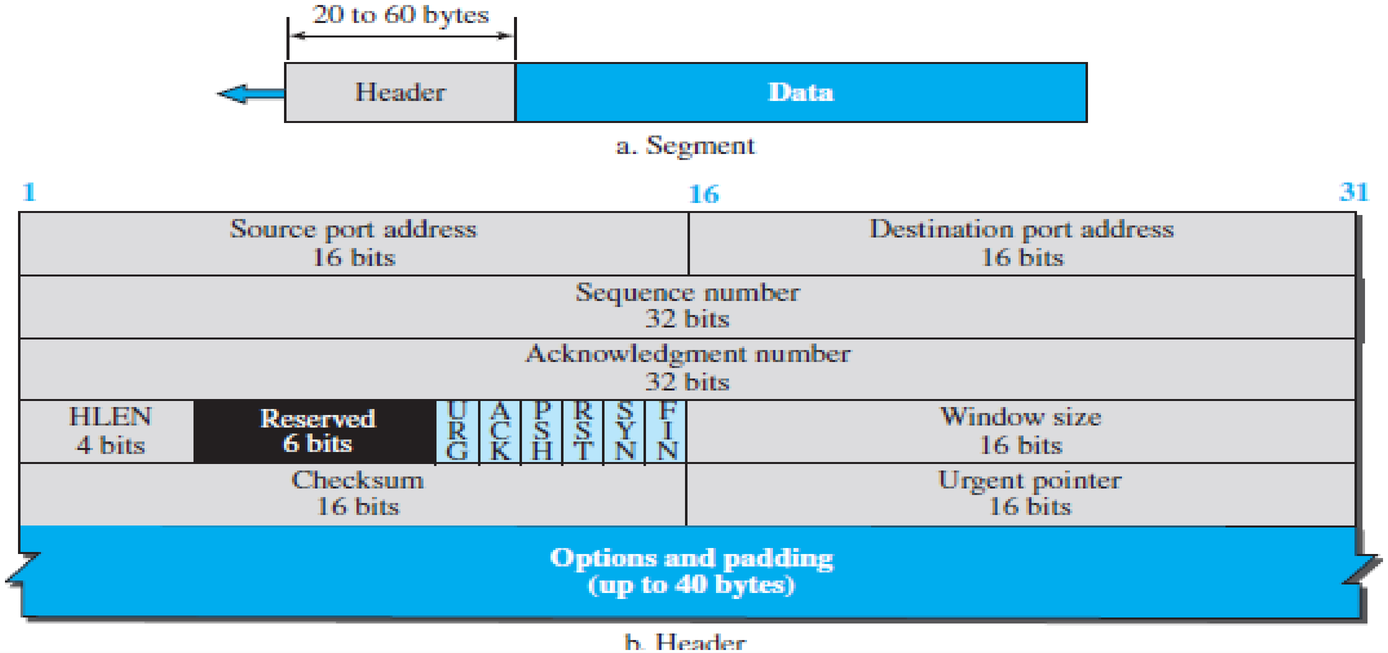
Q Consider a long-lived TCP session with an end-to-end bandwidth of 1 Gbps (= 10^9 bits/second). The session starts with a sequence number of 1234. The minimum time (in seconds, rounded to the closest integer) before this sequence number can be used again is __________. **(GATE-2018) (1 Marks)**

Acknowledgment Number

- This 32-bit field defines the byte number that the receiver of the segment is expecting to receive from the other party. If the receiver of the segment has successfully received byte number x from the other party, it defines x + 1 as the acknowledgment number. Acknowledgment and data can be piggybacked together.

- 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.

- Acknowledgment number can be calculated by subtracting header length of IP and TCP to get the total byte count of the TCP segment and then can find the ack no



a. Segment

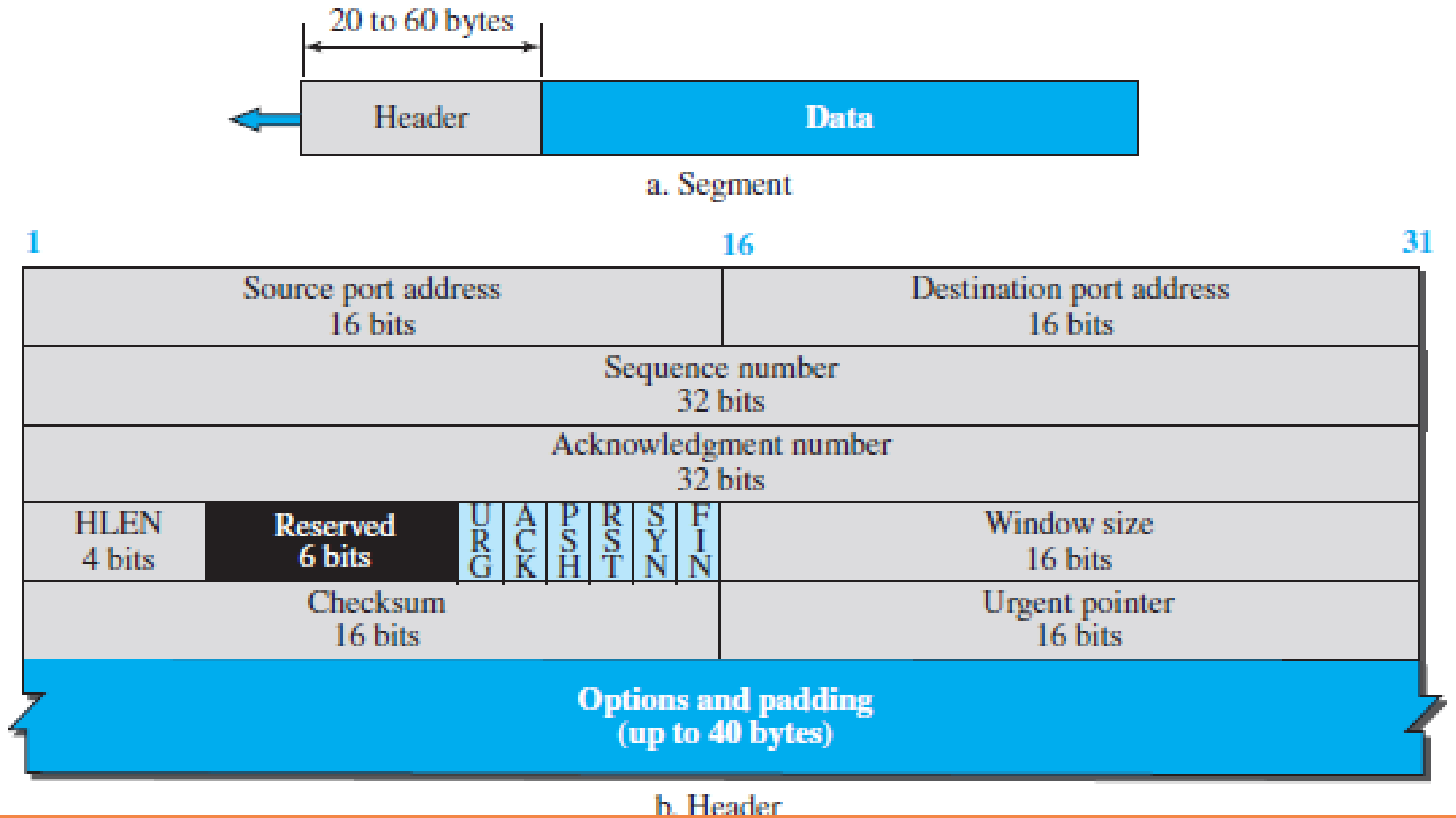
b. Header

Break

Header length

- Header length: This 4-bit field indicates the number of4-byte words in the TCP header. The length of the header can be between 20 and 60 bytes. Therefore, the value of this field can be between 5 (5 x 4 =20) and 15 (15 x 4 =60).
- Concept of Scaling Factor
    - *Header length = Header length field value x 4 bytes*

- Reserved. This is a 6-bit field reserved for future use.



20 to 60 bytes

Header

Data

a. Segment

1

16

31

Source port address 16 bits

Destination port address 16 bits

Sequence number 32 bits

Acknowledgment number 32 bits

HLEN 4 bits

Reserved 6 bits

URG

ACK

PSH

RST

SYN

FIN

Window size 16 bits

Checksum 16 bits

Urgent pointer 16 bits

Options and padding (up to 40 bytes)

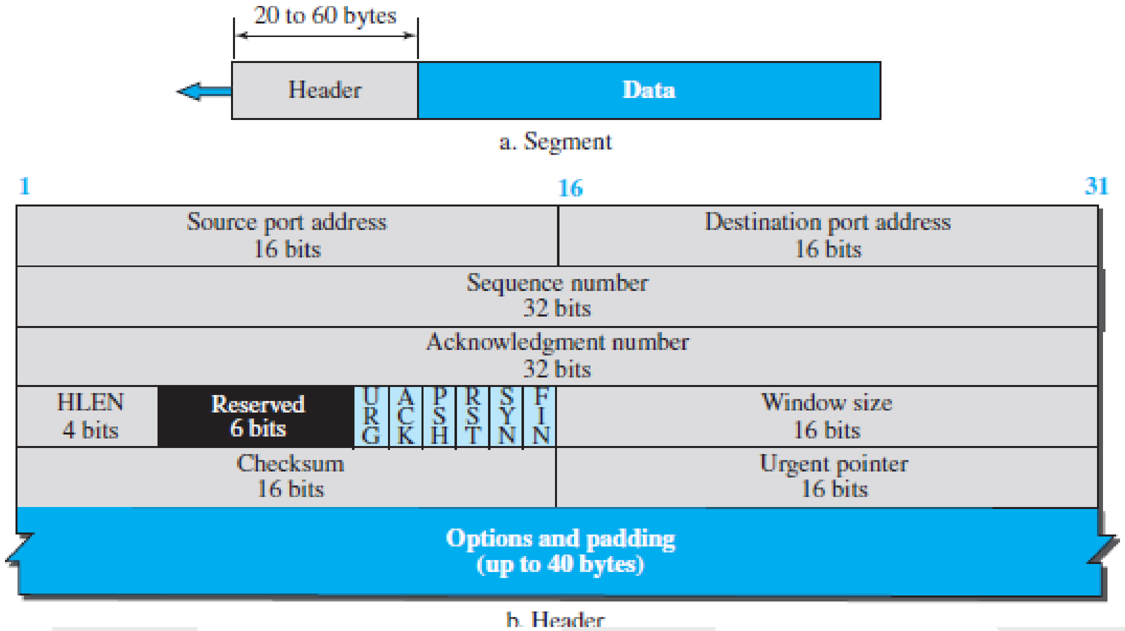
b. Header

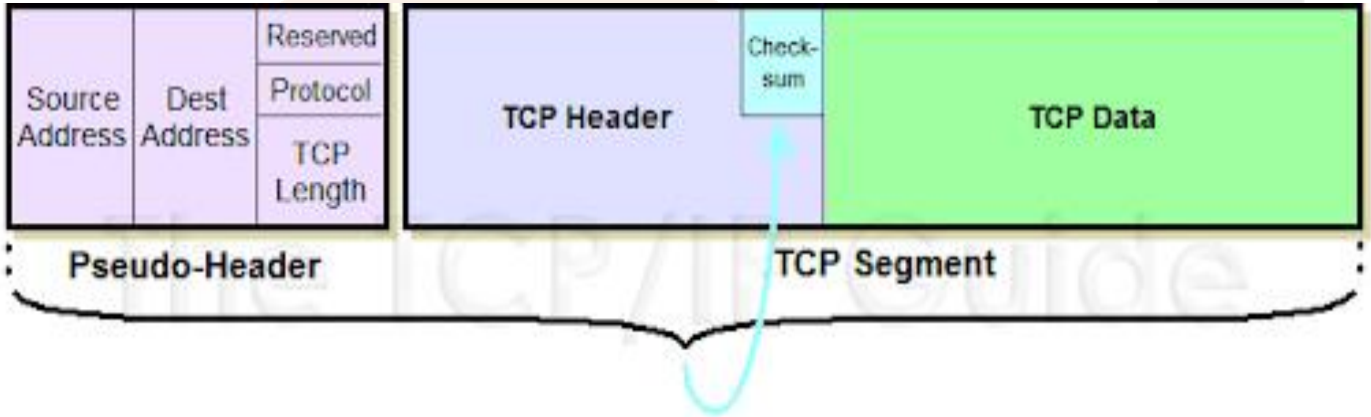
Checksum

- **Checksum:** This 16-bit field contains the checksum.

- While calculation of the checksum for TCP, Entire TCP segment and pseudo header (IP) is considered.

- For the TCP pseudo header, the value for the protocol field is 6.





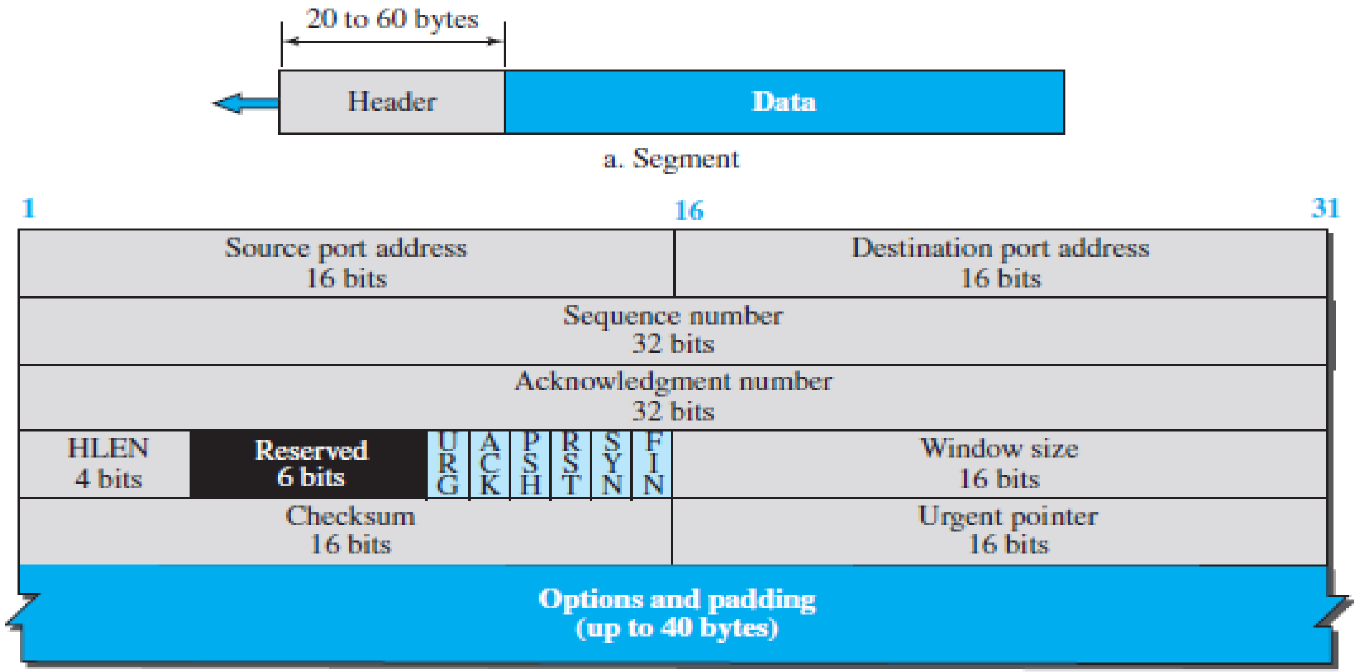
Checksum Calculated Over Pseudo Header and TCP Segment

Window Size/Advertisement window(Flow control)

- Basics idea is a sender should never send what a receiver can not receive

- ***Window size:*** This field defines the size of the window, in bytes, that the receiver have reserved for the incoming data from sender.

- Note that the length of this field is 16 bits, which means that the maximum size of the window is 65,535 bytes. This value is normally referred to as the receiving window and is determined by the receiver. The sender must obey the dictation of the receiver in this case.



20 to 60 bytes

Header | Data

a. Segment

1 | 16 | 31

Source port address 16 bits | Destination port address 16 bits

Sequence number 32 bits

Acknowledgment number 32 bits

HLEN 4 bits | Reserved 6 bits | URG | ACK | PSH | RST | SYN | FIN | Window size 16 bits

Checksum 16 bits | Urgent pointer 16 bits

Options and padding (up to 40 bytes)

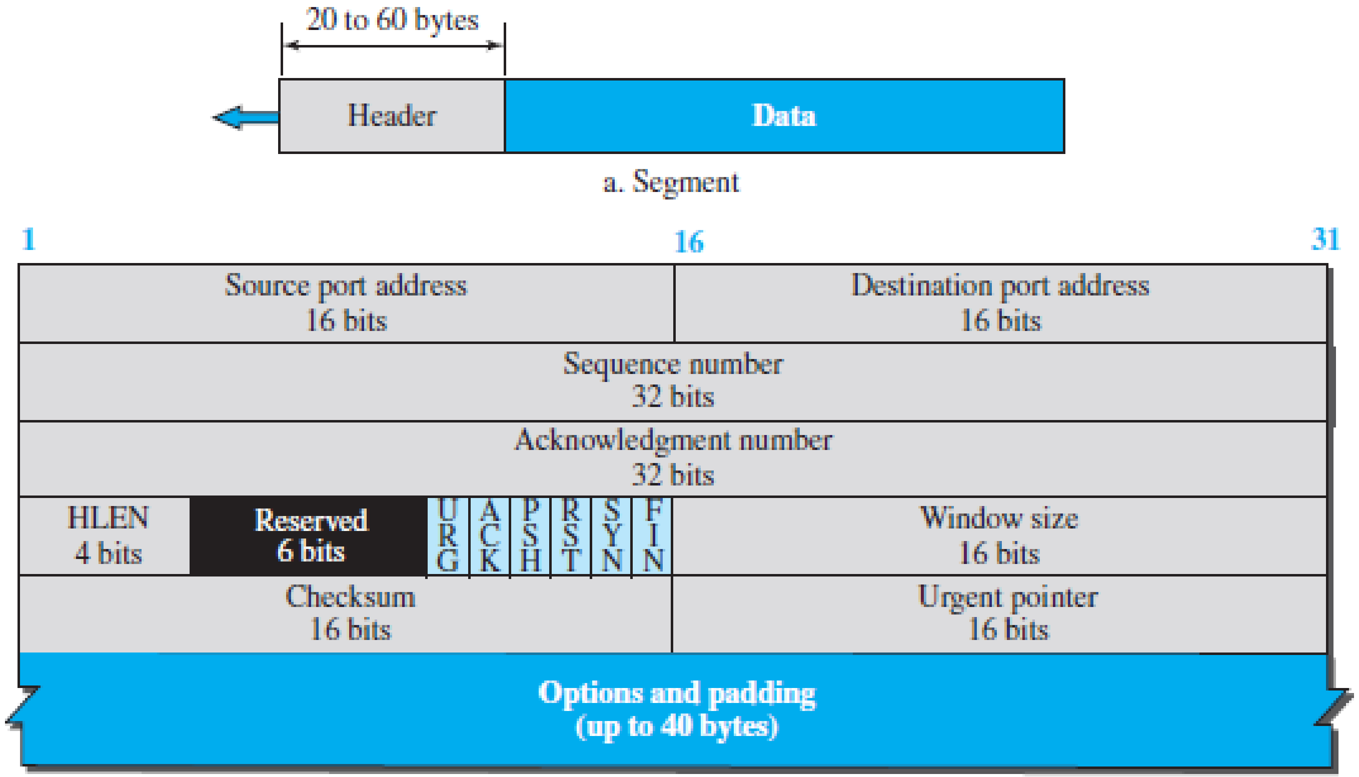
b. Header

Window Size/Advertisement window(Flow control)

- **Persistent Timer –** To deal with a zero-window-size deadlock situation, TCP uses a persistence timer.

- When the sending TCP receives an acknowledgment with a window size of zero, it starts a persistence timer. When the persistence timer goes off, the sending TCP sends a special segment called a probe.

- This segment contains only 1 byte of new data. It has a sequence number, but its sequence number is never acknowledged. The probe causes the receiving TCP to resend the acknowledgment which was lost.

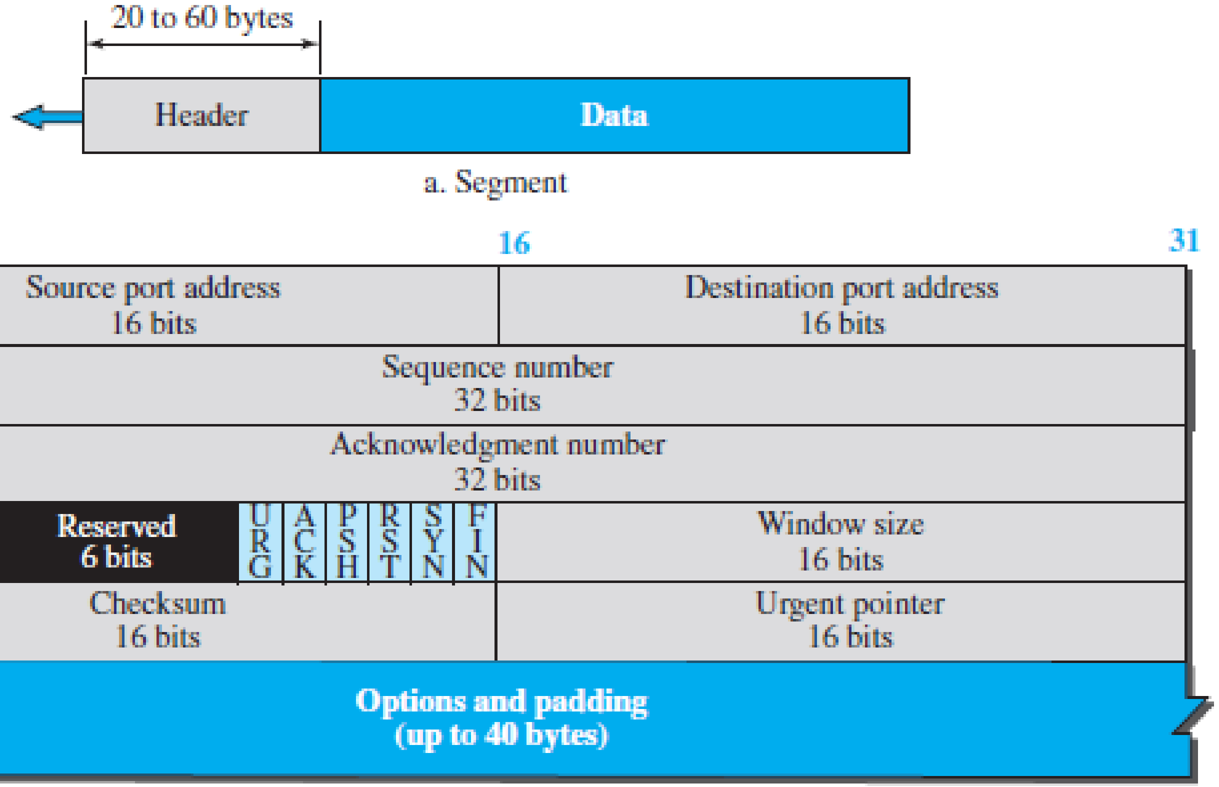


a. Segment

b. Header

Window Size/Advertisement window(Flow control)

- One problem with this idea is window size is only 16 bits long, so very small amount of data can be advertise in todays world context.
- A solution is additional 14 bits can be taken from options so total size become 30 bits or 1GB.

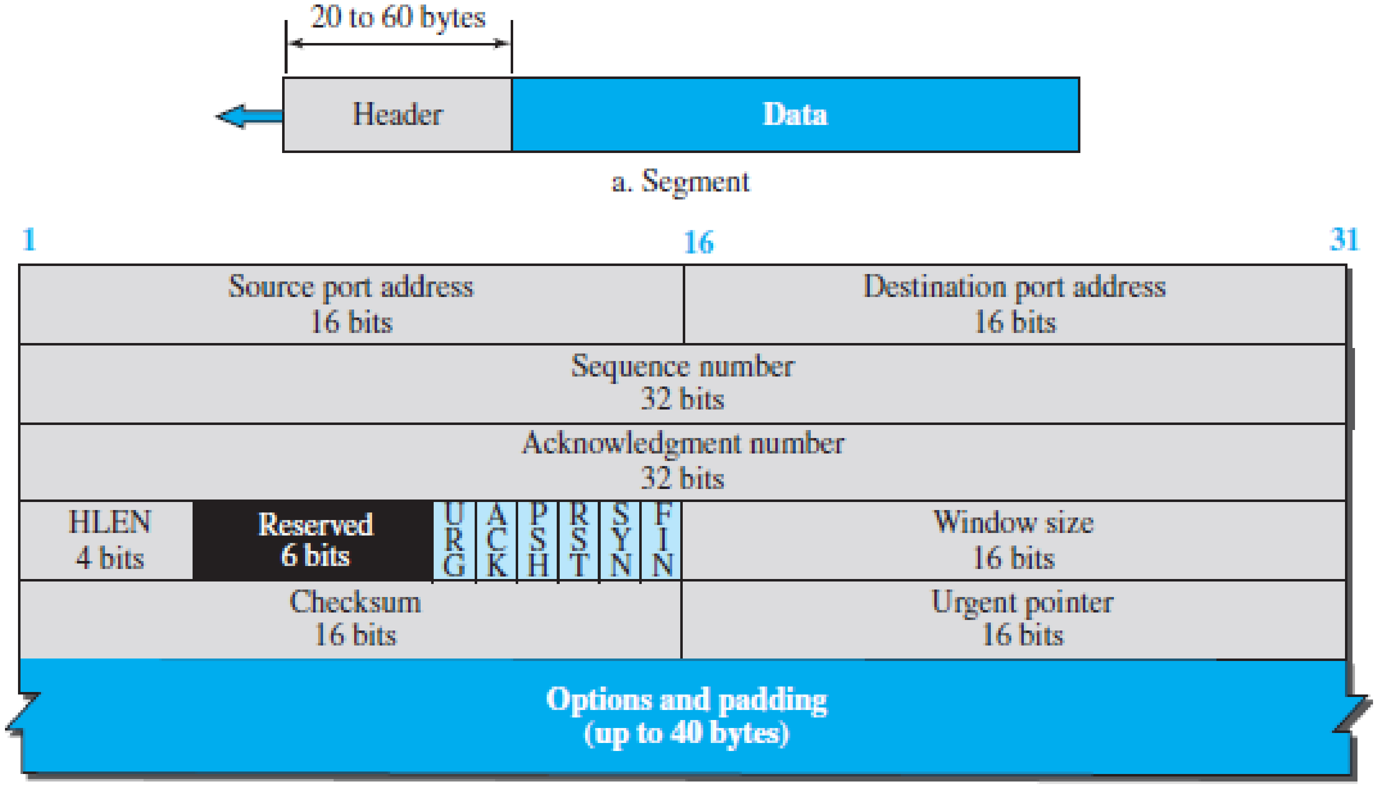


a. Segment

b. Header

Urgent pointer

- **Urgent pointer:** This 16-bit field, which is valid only if the urgent flag is set, is used when the segment contains urgent data. It defines the number that must be added to the sequence number to obtain the number of the last urgent byte in the data section of the segment.



20 to 60 bytes

Header

Data

a. Segment

1

16

31

Source port address
16 bits

Destination port address
16 bits

Sequence number
32 bits

Acknowledgment number
32 bits

HLEN
4 bits

Reserved
6 bits

URG

ACK

PSH

RST

SYN

FIN

Window size
16 bits

Checksum
16 bits

Urgent pointer
16 bits

Options and padding
(up to 40 bytes)

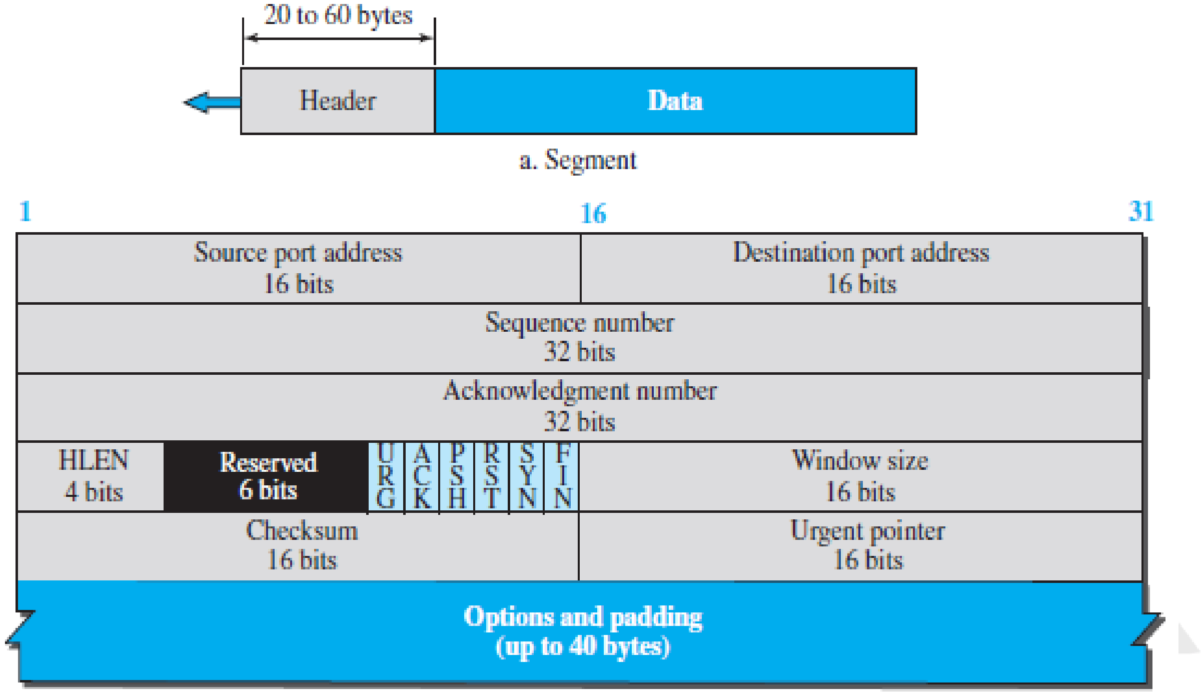
b. Header

Break

Control Flag

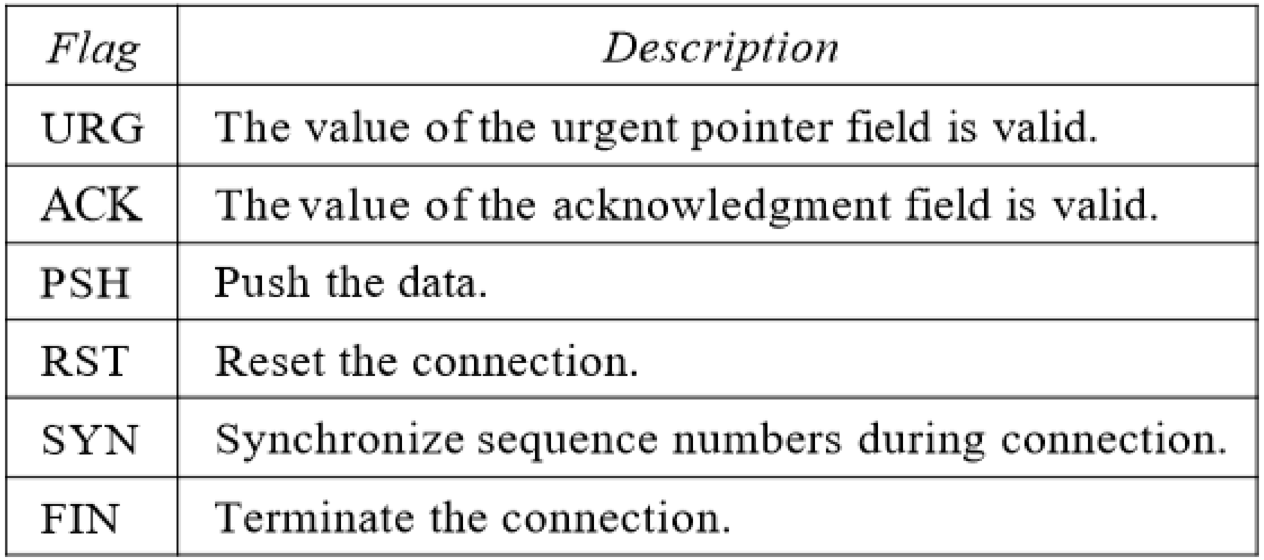
- **Control Flag:** This field defines 6 different control bits or flags. One or more of these bits can be set at a time.

- These bits enable flow control, connection establishment and termination, connection abortion, and the mode of data transfer in TCP.



a. Segment

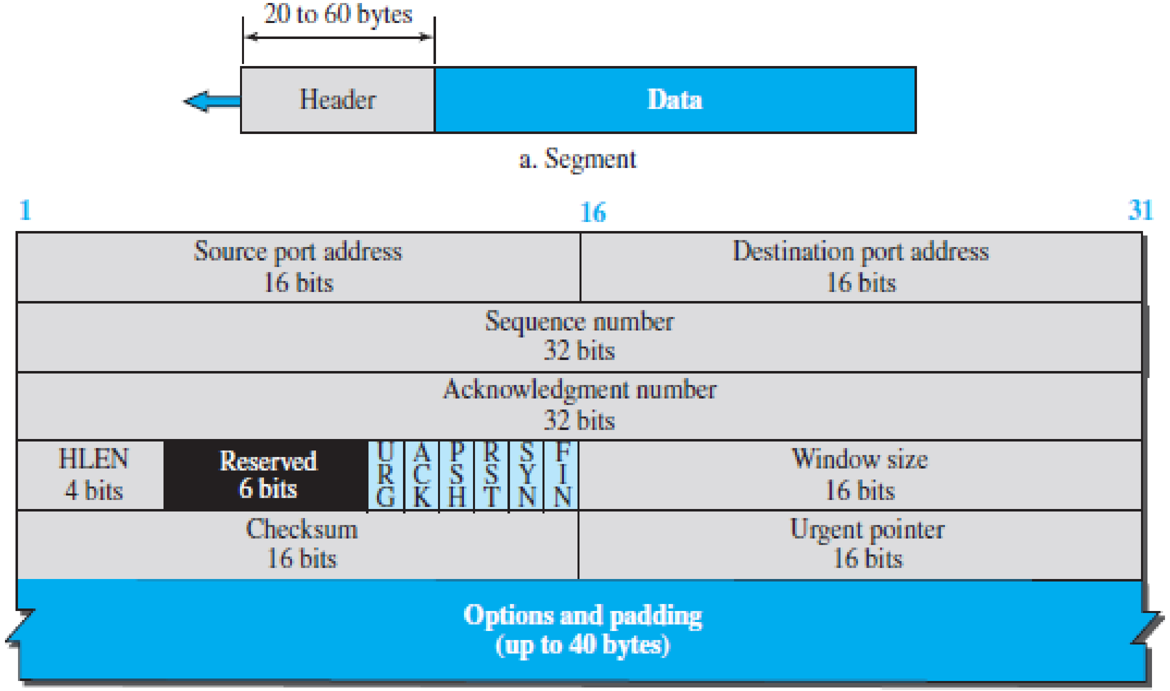
b. Header



Flag	*Description*
URG	The value of the urgent pointer field is valid.
ACK	The value of the acknowledgment field is valid.
PSH	Push the data.
RST	Reset the connection.
SYN	Synchronize sequence numbers during connection.
FIN	Terminate the connection.

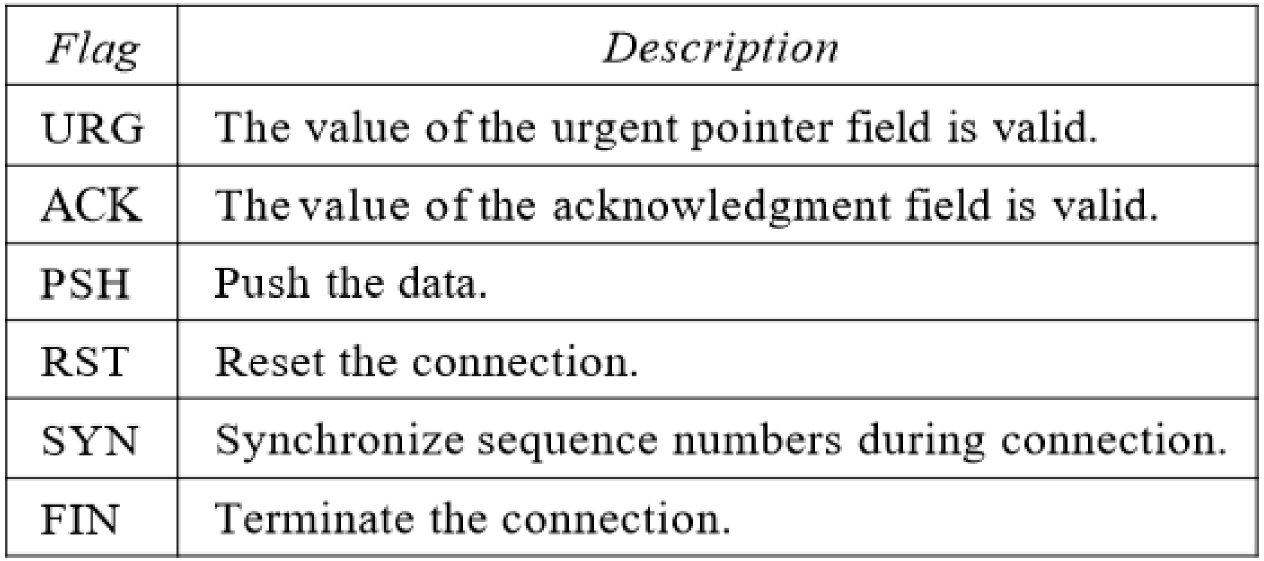
PUSH Flag

- **Push (PSH) –** Transport layer by default waits for some time for application layer to send enough data equal to maximum segment size so that the number of packets transmitted on network minimizes which is not desirable by some application like interactive applications(chatting).
- Similarly transport layer at receiver end buffers packets and transmit to application layer if it meets certain criteria. This problem is solved by using PSH. Transport layer sets PSH = 1 and immediately sends the segment to network layer as soon as it receives signal from application layer.
- Receiver transport layer, on seeing PSH = 1 immediately forwards the data to application layer. In general, it tells the receiver to process these packets as they are received instead of buffering them.



a. Segment

b. Header

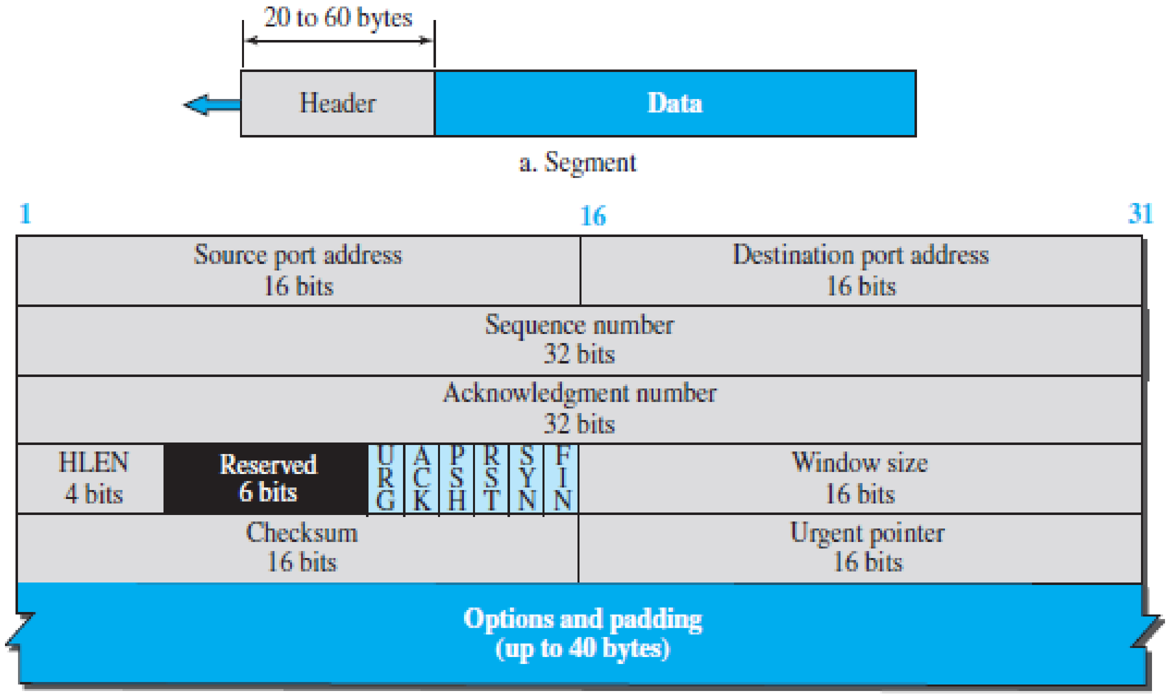


Flag	*Description*
URG	The value of the urgent pointer field is valid.
ACK	The value of the acknowledgment field is valid.
PSH	Push the data.
RST	Reset the connection.
SYN	Synchronize sequence numbers during connection.
FIN	Terminate the connection.

RST Flag

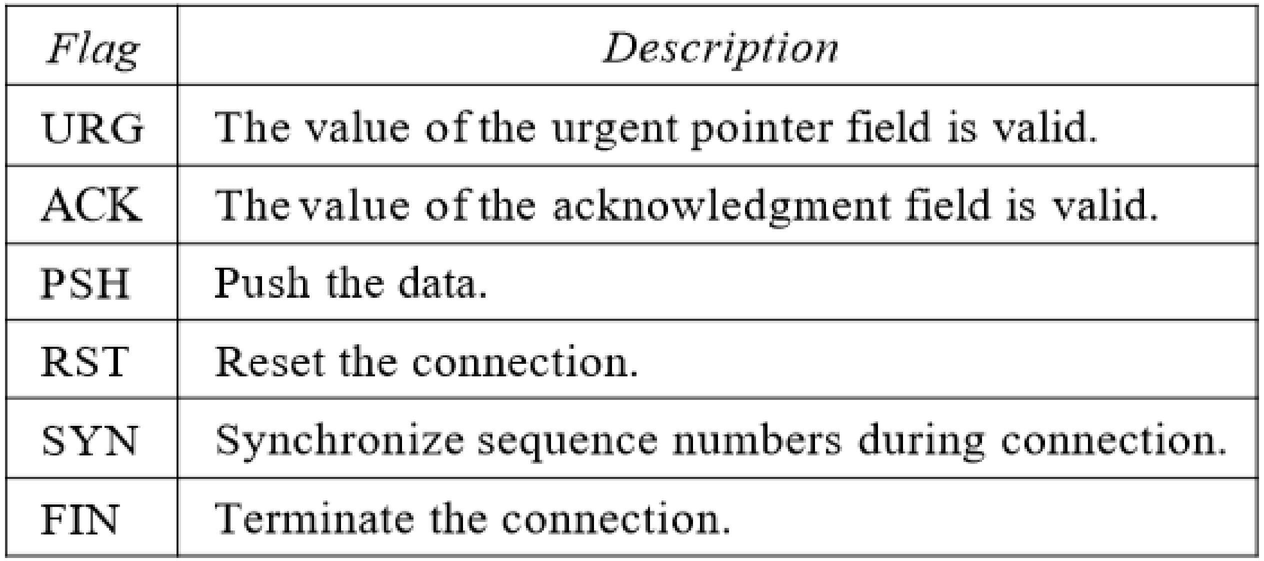
- **Reset (RST) –** It is used to terminate the connection if the sender or receiver feels something is wrong with the TCP connection or that the conversation should not exist.

- It can get send from receiver side when packet is send to particular host that was not expecting it.



a. Segment

b. Header

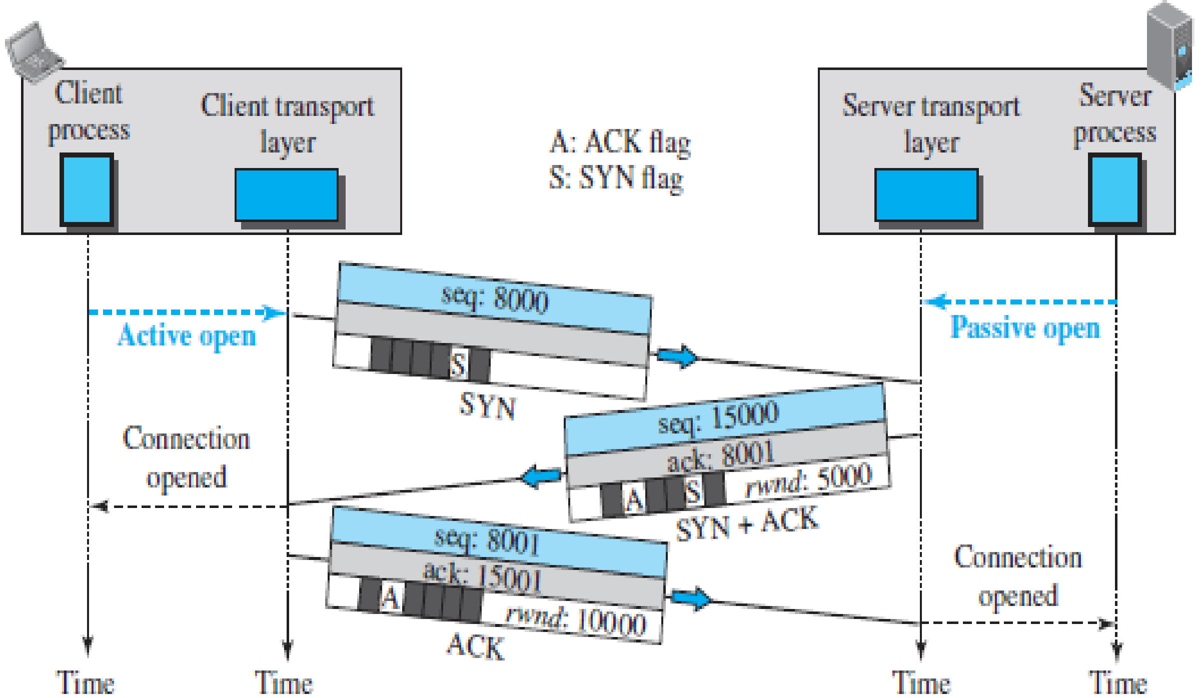


Flag	*Description*
URG	The value of the urgent pointer field is valid.
ACK	The value of the acknowledgment field is valid.
PSH	Push the data.
RST	Reset the connection.
SYN	Synchronize sequence numbers during connection.
FIN	Terminate the connection.

Break

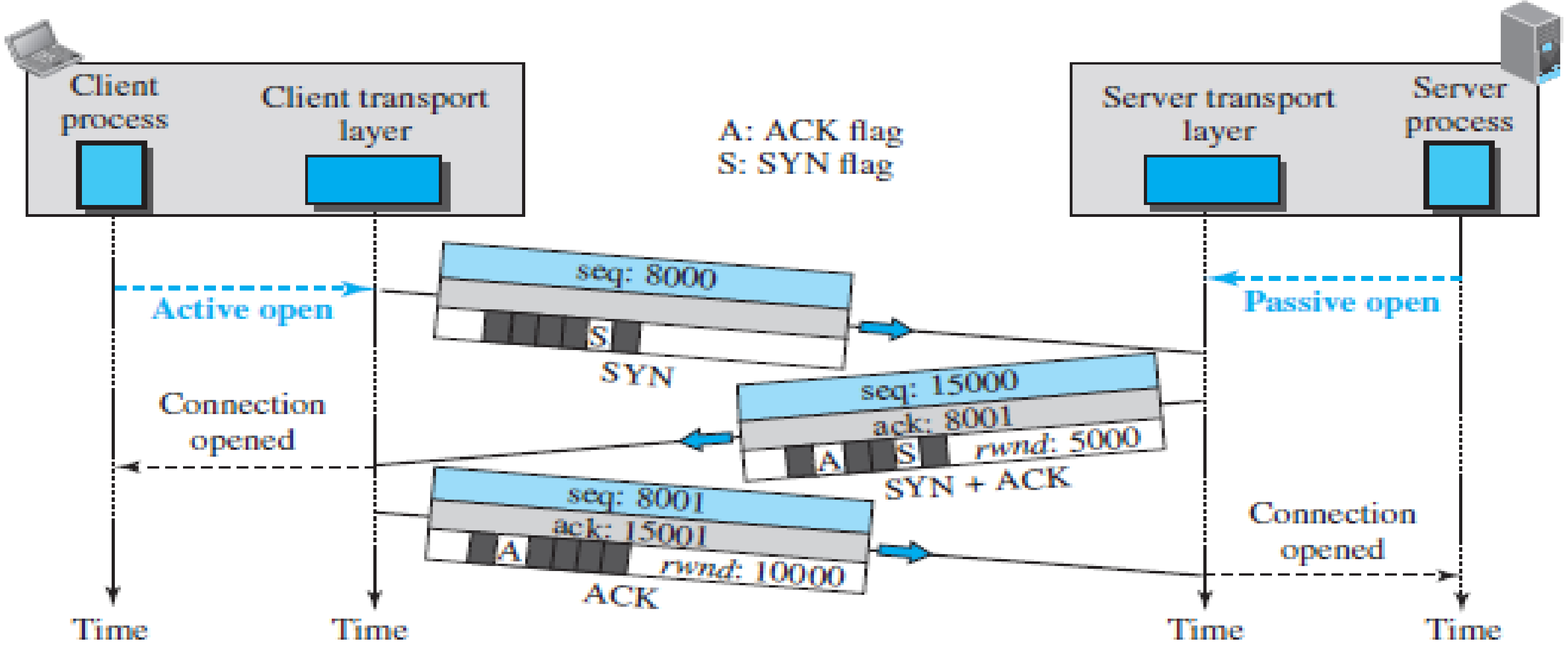
A TCP Connection

- The connection establishment in TCP is called ***three-way handshaking.***
- 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.

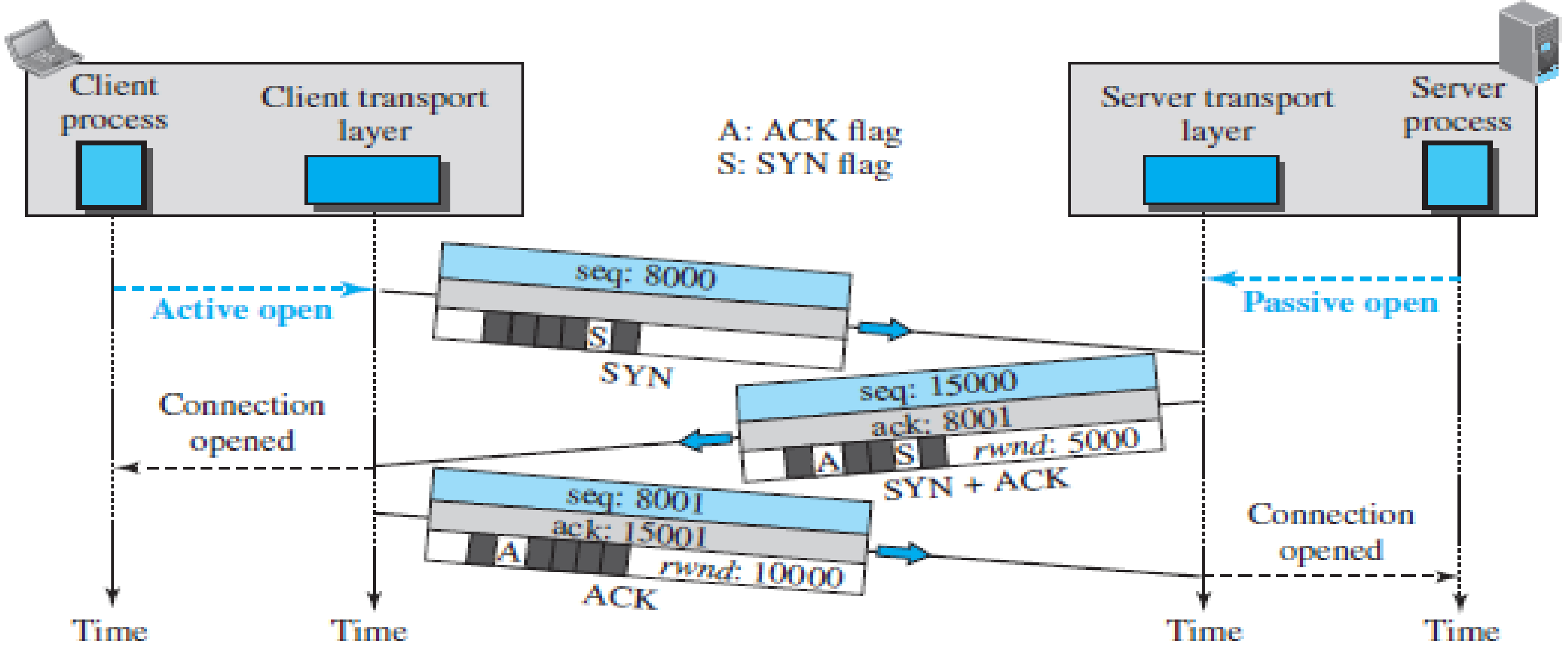


Connection Establishment (Three-way handshaking)

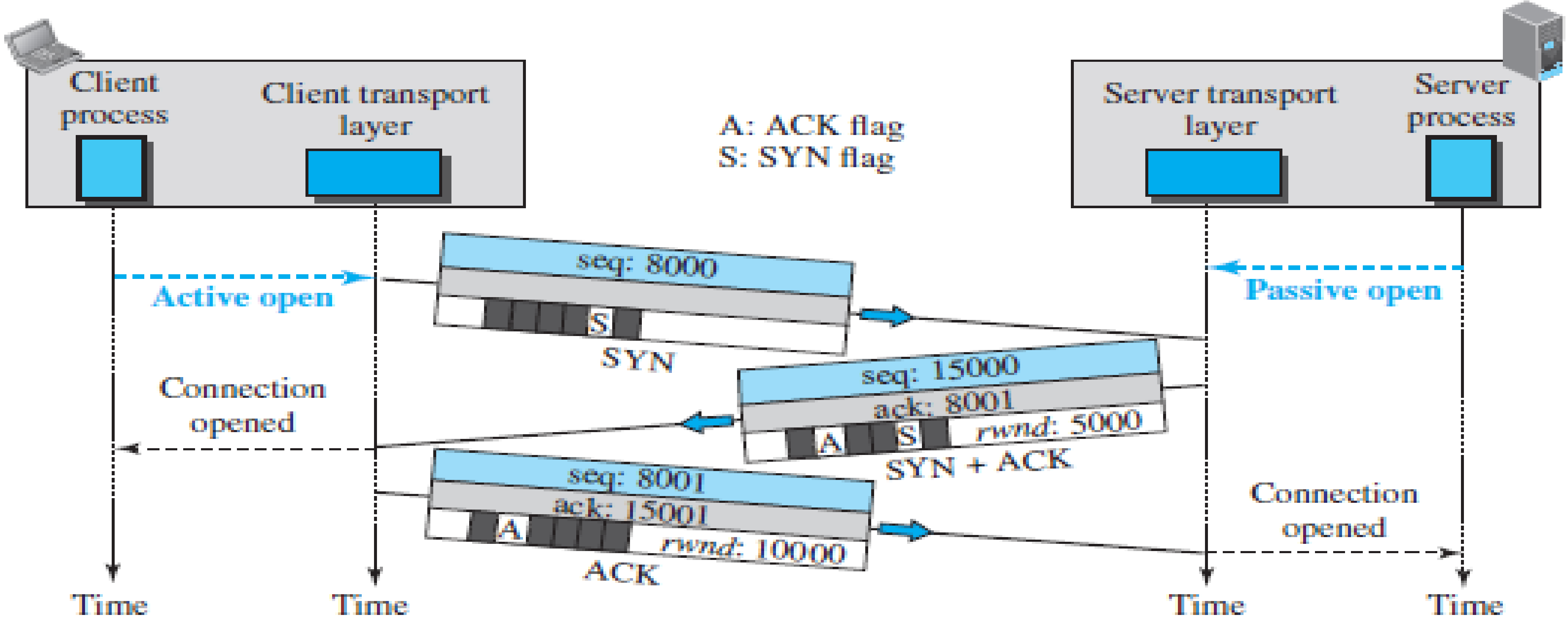
- The client sends the first segment, a SYN segment, in which only the SYN flag is set. This segment is for synchronization of sequence numbers.
  - Client sends the ***initial sequence number (ISN).***
  - This segment does not contain an acknowledgment number
  - SYN segment is a control segment and carries no data. However, it consumes one sequence number because it needs to be acknowledged.



- The server sends the second segment, a SYN + ACK segment with two flag bits set as: SYN and ACK.
  - It is a SYN segment for communication in the other direction.
  - The server uses this segment to initialize a sequence number for numbering the bytes sent from the server to the client.
  - The server also 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.
  - A SYN + ACK segment cannot carry data, but it does consume one sequence number.



- 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.
- ACK segment does not consume any sequence numbers if it does not carry data.
- After connection is established, bidirectional data transfer can take place.



Q.4 Consider the three-way handshake mechanism followed during TCP connection establishment between hosts P and Q. Let X and Y be two random 32-bit starting sequence numbers chosen by P and Q respectively. Suppose P sends a TCP connection request message to Q with a TCP segment having SYN bit = 1. SEQ number = X, and ACK bit = 0. Suppose Q accepts the connection request. Which one of the following choices represents the information present in the TCP segment header that is sent by Q to P? **(GATE-2021)**

(a) SYN bit = 1, SEQ number = Y, ACK bit = 1, ACK number = $X + 1$, FIN bit = 0

(b) SYN bit = 1, SEQ number = Y, ACK bit = 1, ACK number = X, FIN bit = 0

(c) SYN bit = 0, SEQ number = $X + 1$, ACK bit = 0, ACK number = Y, FIN bit = 1

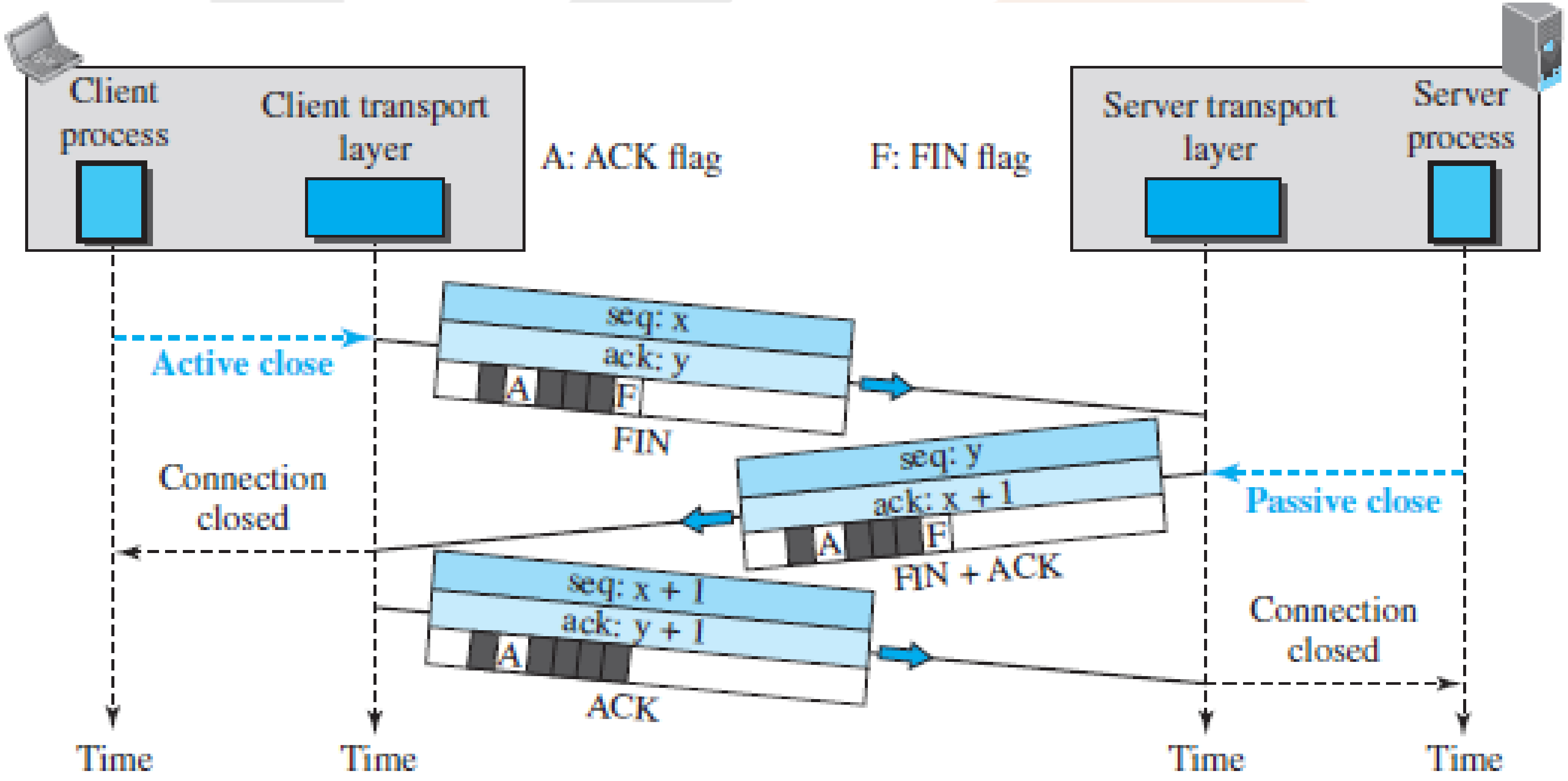
(d) SYN bit = 1, SEQ number = $X + 1$, ACK bit = 0, ACK number = Y, FIN bit = 0

Ans. (a)

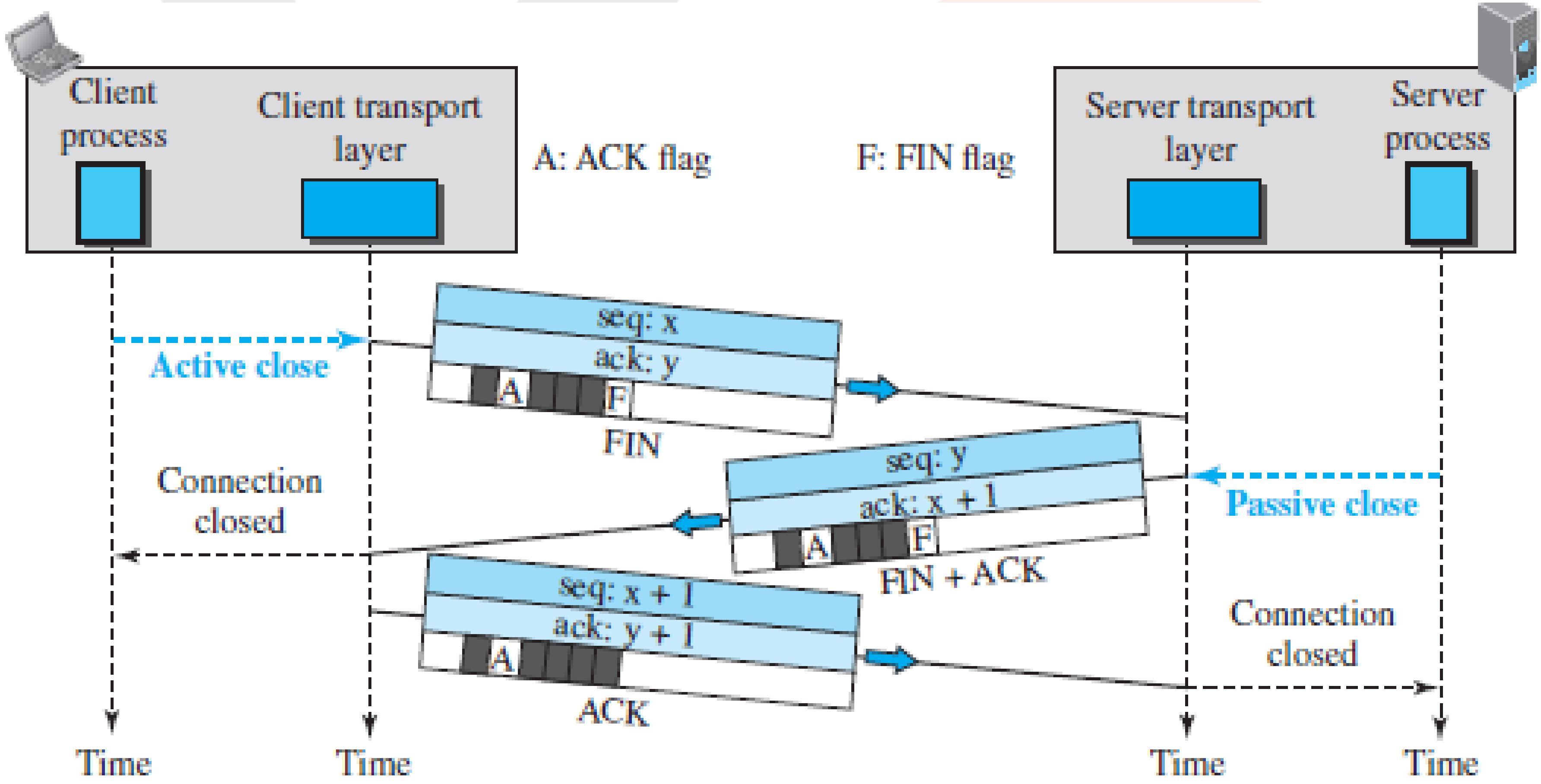
Break

Connection Termination (Three-way handshaking)

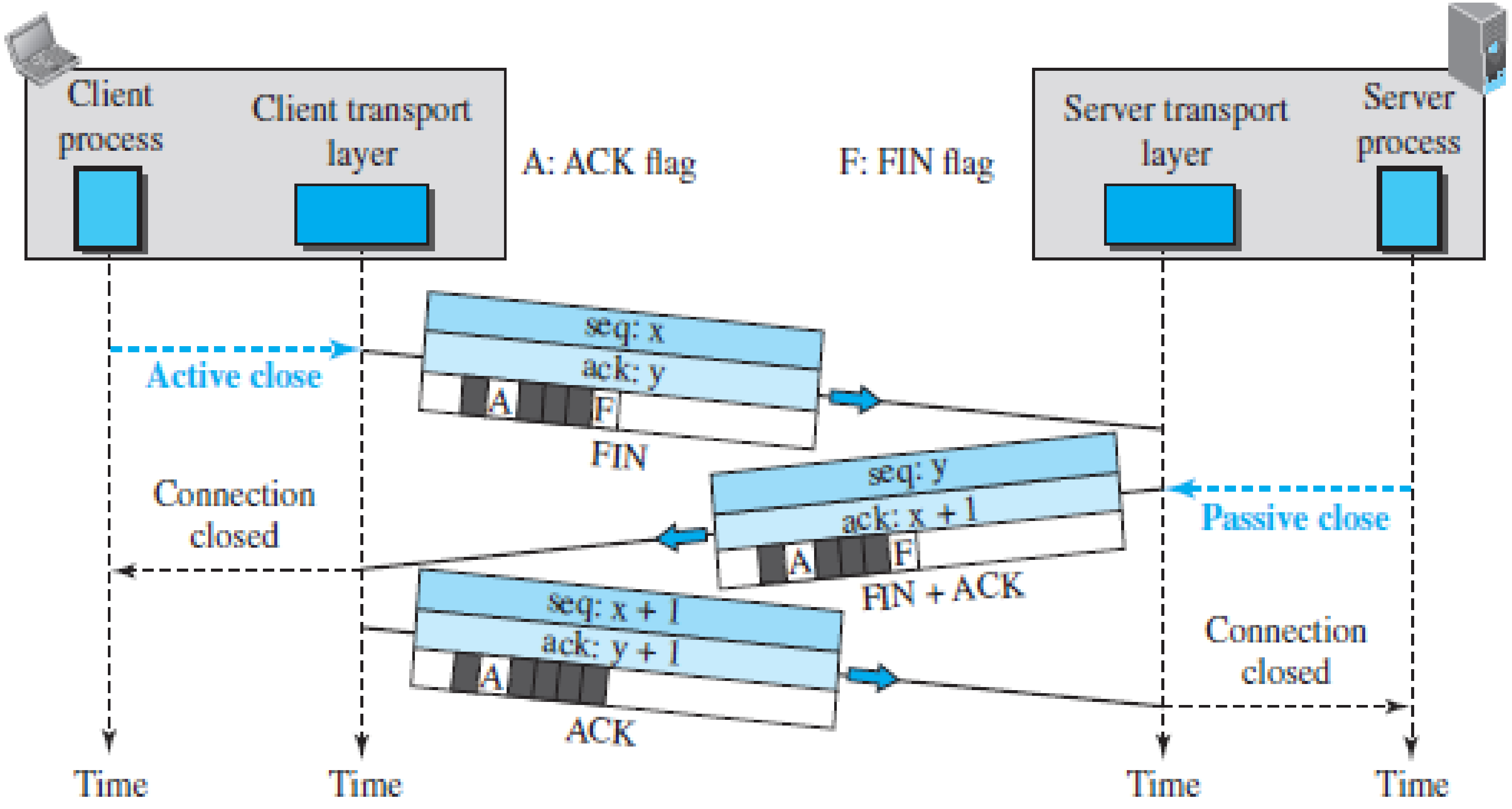
- Either of the two parties involved in exchanging data (client or server) can close the connection
- In this situation, the client TCP, after receiving a close command from the client process, sends the first segment, a FIN segment in which the FIN flag is set.
  - The FIN segment consumes one sequence number if it does not carry data.



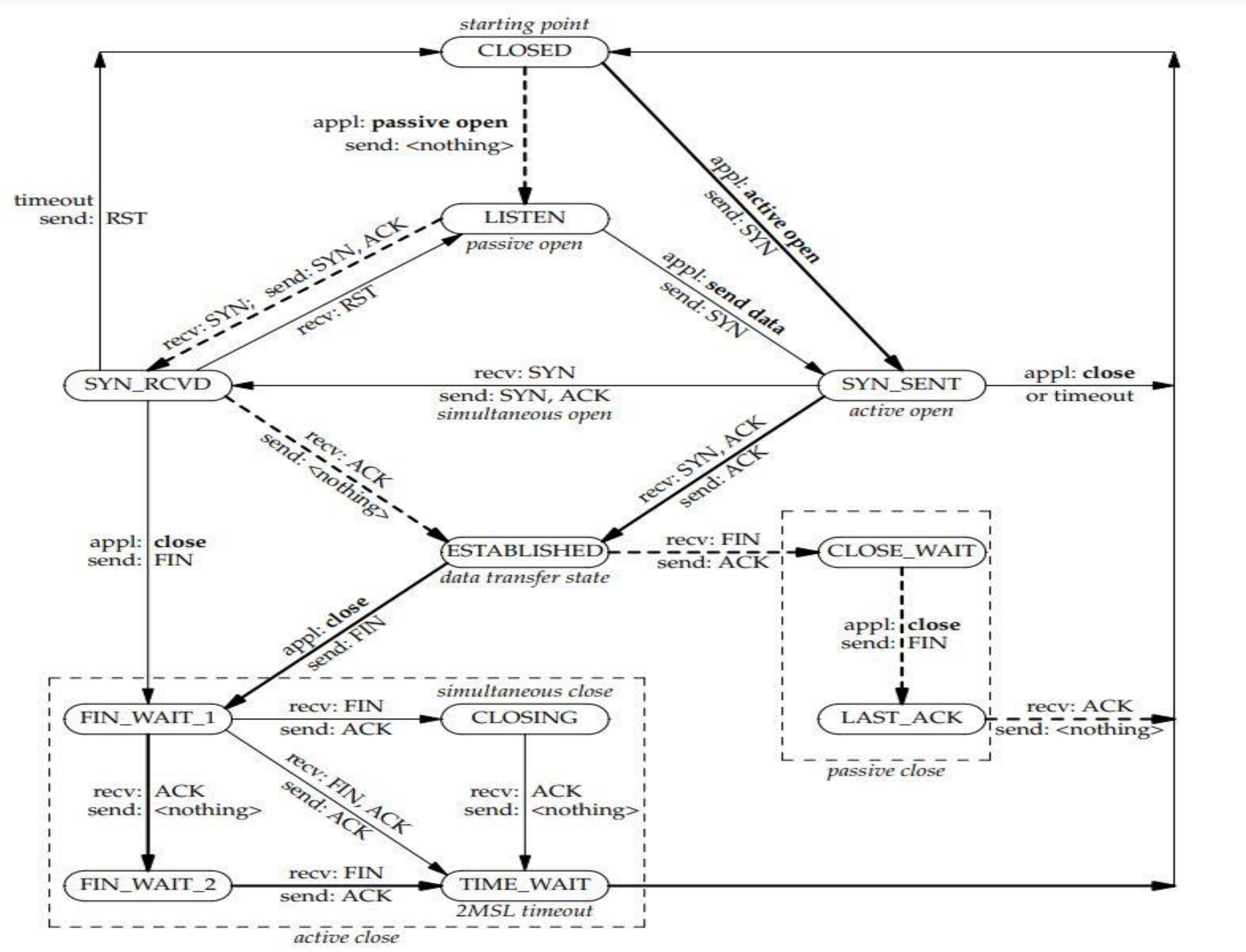
- The server TCP, after receiving the FIN segment, informs its process of the situation and sends the second segment, a FIN + ACK segment, to confirm the receipt of the FIN segment from the client and at the same time to announce the closing of the connection in the other direction.
  - If it does not carry data, it consumes only one sequence number because it needs to be acknowledged.



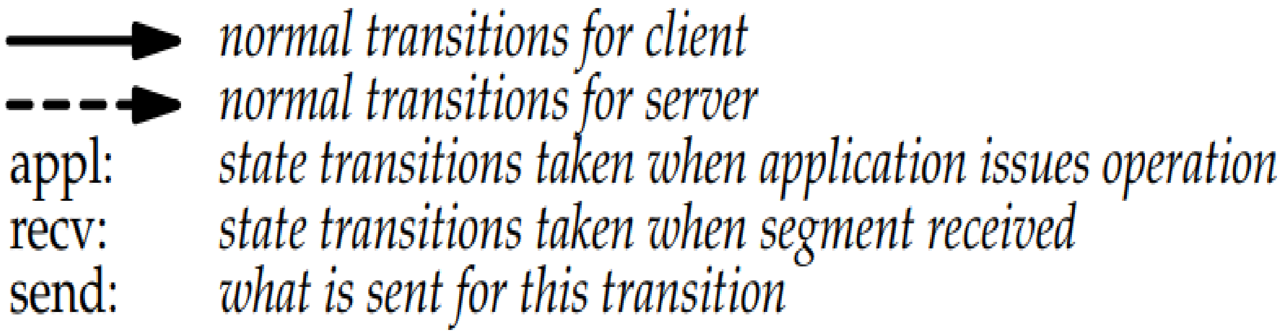
- The client TCP sends the last segment, an ACK segment, to confirm the receipt of the FIN segment from the TCP server.
  - This segment cannot carry data and consumes no sequence numbers.



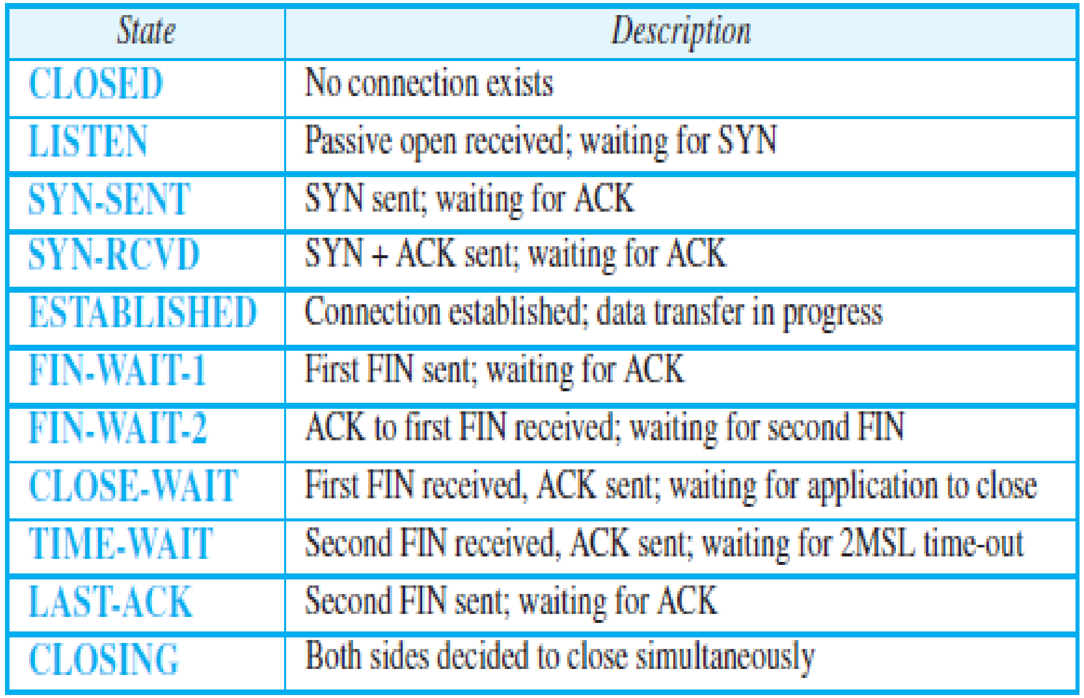
Break


starting point
CLOSED
appl: passive open
send: <nothing>
appl: active open
send: SYN
timeout
send: RST
LISTEN
passive open
recv: SYN; send: SYN, ACK
recv: RST
appl: send data
send: SYN
SYN_RCVD
recv: SYN
send: SYN, ACK
simultaneous open
SYN_SENT
active open
appl: close
or timeout
recv: ACK
send: <nothing>
recv: SYN, ACK
send: ACK
appl: close
send: FIN
ESTABLISHED
data transfer state
recv: FIN
send: ACK
CLOSE_WAIT
appl: close
send: FIN
LAST_ACK
recv: ACK
send: <nothing>
passive close
appl: close
send: FIN
simultaneous close
FIN_WAIT_1
recv: FIN
send: ACK
CLOSING
recv: FIN, ACK
send: ACK
recv: ACK
send: <nothing>
recv: ACK
send: <nothing>
FIN_WAIT_2
recv: FIN
send: ACK
TIME_WAIT
2MSL timeout
active close

States for TCP



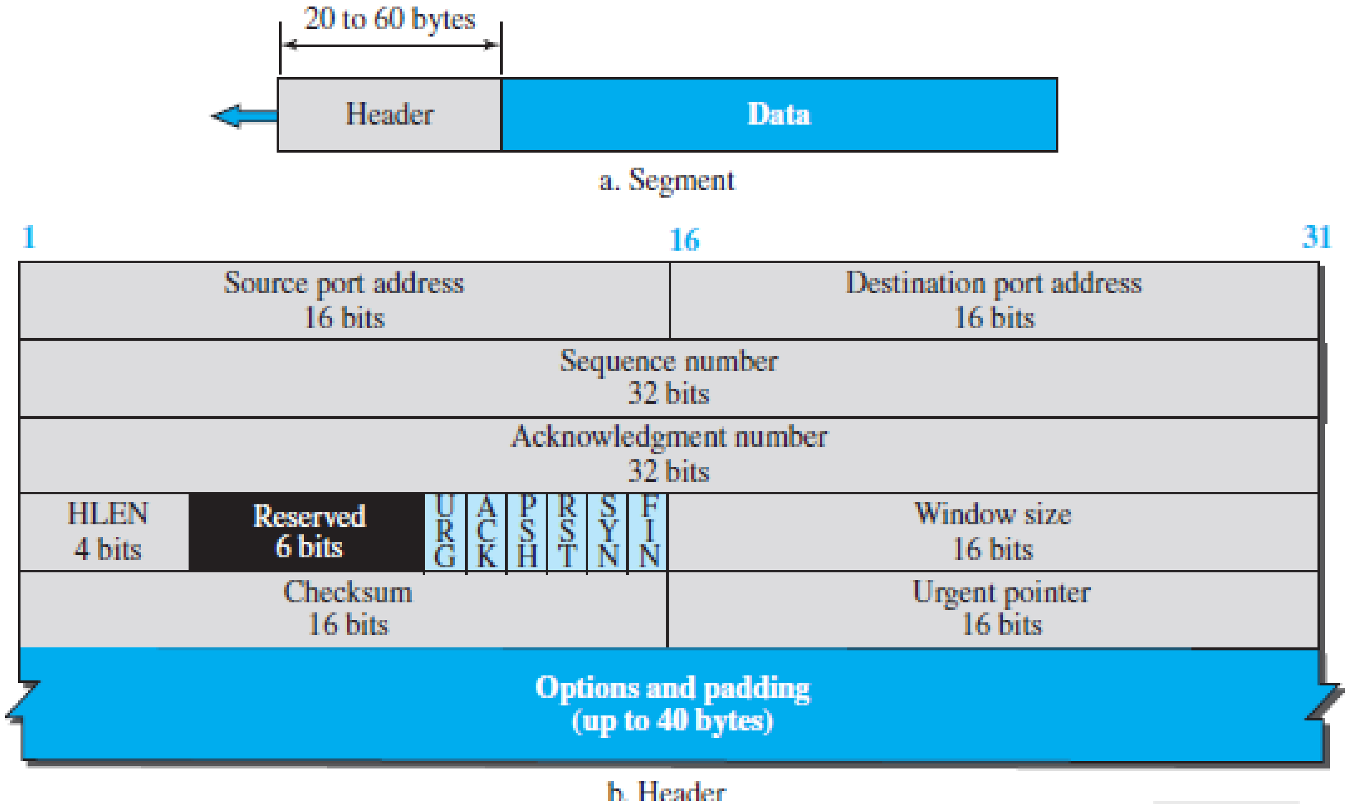
⟶	*normal transitions for client*
⇢	*normal transitions for server*
appl:	*state transitions taken when application issues operation*
recv:	*state transitions taken when segment received*
send:	*what is sent for this transition*



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

Options

- There can be up to 40 bytes of optional information in the TCP header.
  - Time Stamp
  - Window Size Extension
  - Parameter negotiation
    - MSS etc
  - Padding



a. Segment

<table>
<tr><td colspan="8">1 … 16</td><td>31</td></tr>
<tr><td colspan="8">Source port address
16 bits</td><td>Destination port address
16 bits</td></tr>
<tr><td colspan="9">Sequence number
32 bits</td></tr>
<tr><td colspan="9">Acknowledgment number
32 bits</td></tr>
<tr><td>HLEN
4 bits</td><td>Reserved
6 bits</td><td>URG</td><td>ACK</td><td>PSH</td><td>RST</td><td>SYN</td><td>FIN</td><td>Window size
16 bits</td></tr>
<tr><td colspan="8">Checksum
16 bits</td><td>Urgent pointer
16 bits</td></tr>
<tr><td colspan="9">Options and padding
(up to 40 bytes)</td></tr>
</table>

b. Header

Q Suppose two hosts use a TCP connection to transfer a large file. Which of the following statements is/are **False** with respect to the TCP connection? **(Gate-2015) (1 Marks)**

1. If the sequence number of a segment is m, then the sequence number of the subsequent segment is always m+1.
2. If the estimated round-trip time at any given point of time is t sec, the value of the retransmission timeout is always set to greater than or equal to t sec.
3. The size of the advertised window never changes during the course of the TCP connection.
4. The number of unacknowledged bytes at the sender is always less than or equal to the advertised window

(A) 3 only **(B)** 1 and 3 only **(C)** 1 and 4 only **(D)** 2 and 4 only

Break

SYN Flooding Attack

- When one or more malicious attackers send a large number of SYN segments to a server pretending that each of them is coming from a different client by faking the source IP addresses in the datagrams.
- The server allocates the necessary resources, such as creating transfer control block (TCB) tables and setting timers.
- TCP server then sends the SYN + ACK segments to the fake clients, which are lost.
- The server waits for the third leg of the handshaking process and resources are allocated without being used.
- During this short period of time, if the number of SYN segments is large, the server eventually runs out of resources and may be unable to accept connection requests from valid clients.
- SYN flooding attack belongs to denial of service attack.

TCP Retransmission

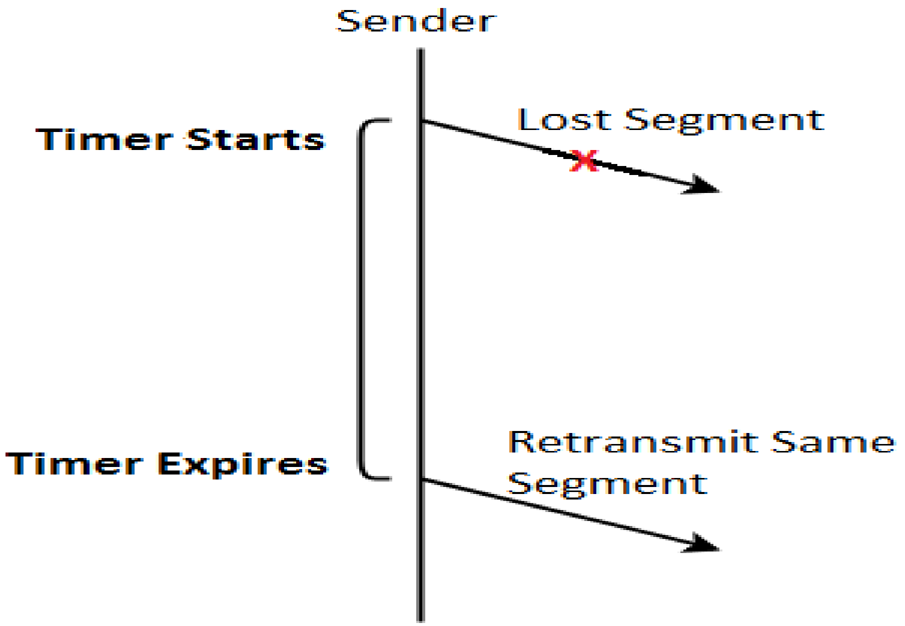
- After establishing the connection, Sender starts transmitting TCP segments to the receiver. A TCP segment sent by the sender may get lost on the way before reaching the receiver.

- This causes the receiver to send the acknowledgement with same ACK number to the sender. As a result, sender retransmits the same segment to the receiver. This is called as **TCP retransmission**.

- Sender discovers that the TCP segment is lost when
  - Either Time Out Timer expire or it receives three duplicate acknowledgements

Retransmission after Time out Timer Expiry

- Each time sender transmits a TCP segment to the receiver, it starts a Time Out Timer. Following two cases are possible
  - Sender receives an acknowledgement for the sent segment before the timer goes off. In this case, sender stops the timer.
  - Sender does not receive any acknowledgement for the sent segment and the timer goes off. In this case, sender assumes that the sent segment is lost. Sender retransmits the same segment to the receiver and resets the timer.



Retransmission After Receiving 3 Duplicate Acknowledgements/ Early Retransmission

- Consider sender receives three duplicate acknowledgements for a TCP segment sent by it. Then, sender assumes that the corresponding segment is lost.

- So, sender retransmits the same segment without waiting for its time out timer to expire. This is known as ***early retransmission*** or Fast retransmission.

Q Consider Sender sends 5 TCP segments to the receiver. The second TCP segment gets lost before reaching the receiver. The sequence of steps that will take place are

On receiving segment-1, receiver sends acknowledgement asking for segment-2 next. (Original ACK)

On receiving segment-3, receiver sends acknowledgement asking for segment-2 next. (1st duplicate ACK)

On receiving segment-4, receiver sends acknowledgement asking for segment-2 next. (2nd duplicate ACK)

On receiving segment-5, receiver sends acknowledgement asking for segment-2 next. (3rd duplicate ACK)

Now, Sender receives 3 duplicate acknowledgements for segment-2 in total. So, sender assumes that the segment-2 is lost. So, it retransmits segment-2 without waiting for its timer to go off.

Points to Note

- In case time out timer expires before receiving the acknowledgement for a TCP segment, then there is a strong possibility of congestion in the network.

- Retransmission on receiving 3 duplicate acknowledgements is a way to improve the performance over retransmission on time out.

- TCP uses SR (80%) and GBN (20%) both, as Ws =Wr (SR) out of order packets will be accepted and in GBN use cumulative acknowledgement

- Question is why only 3 duplicate ack, experimentally it is found out that this works best.

Q Consider a TCP client and a TCP server running on two different machines. After completing data transfer, the TCP client calls *close* to terminate the connection and a FIN segment is sent to the TCP server. Server-side TCP responds by sending an ACK, which is received by the client-side TCP. As per the TCP connection state diagram (RFC 793), in which state does the client-side TCP connection wait for the FIN from the server-side TCP? **(GATE-2017) (1 Marks)**

(a) LAST-ACK

(b) TIME-WAIT

(c) FIN-WAIT-1

(d) FIN-WAIT-2

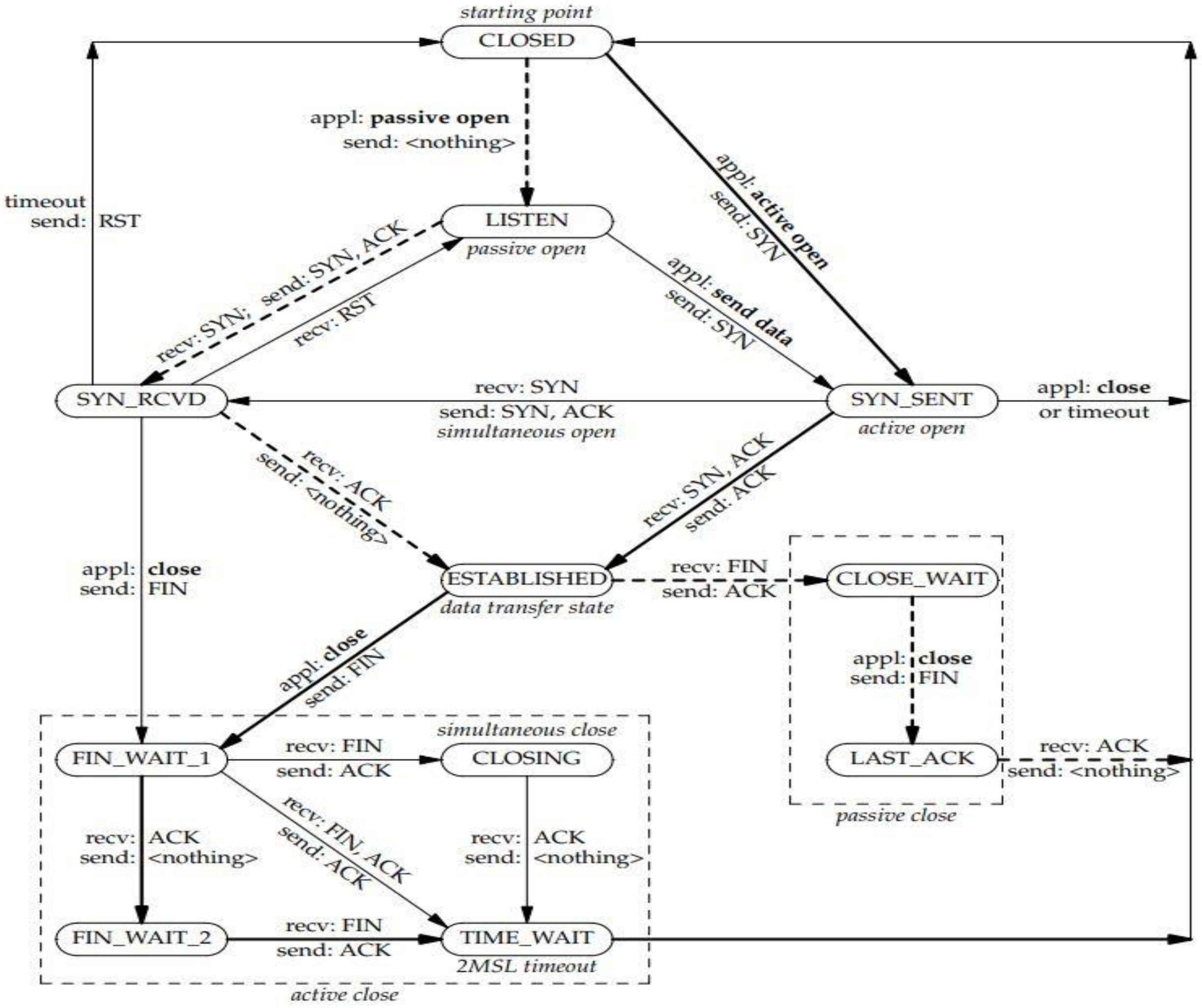
Q Consider a TCP client and a TCP server running on two different machines. After completing data transfer, the TCP client calls *close* to terminate the connection and a FIN segment is sent to the TCP server. Server-side TCP responds by sending an ACK, which is received by the client-side TCP. As per the TCP connection state diagram (RFC 793), in which state does the client-side TCP connection wait for the FIN from the server-side TCP? **(GATE-2017) (1 Marks)**

(a) LAST-ACK

(b) TIME-WAIT

(c) FIN-WAIT-1

(d) FIN-WAIT-2



Q Assume that the bandwidth for a TCP connection is 10,48,560 bits/sec. Let α be the value of RTT in milliseconds (rounded off to the nearest integer) after which the TCP window scale option is needed. Let β be the maximum possible window size with window scale option. Then the values of α and β are. **(Gate-2015) (2 Marks)**

(A) 63 milliseconds, 65535 × 2^{14}

(B) 63 milliseconds, 65535 × 2^{16}

(C) 500 milliseconds, 65535 × 2^{14}

(D) 500 milliseconds, 65535 × 2^{16}

Q Consider the following statements. **(Gate-2015) (1 Marks)**

I. TCP connections are full duplex.

II. TCP has no option for selective acknowledgment

III. TCP connections are message streams.

(A) Only I is correct

(B) Only I and II are correct

(C) Only II and III are correct

(D) All of I, II and III are correct

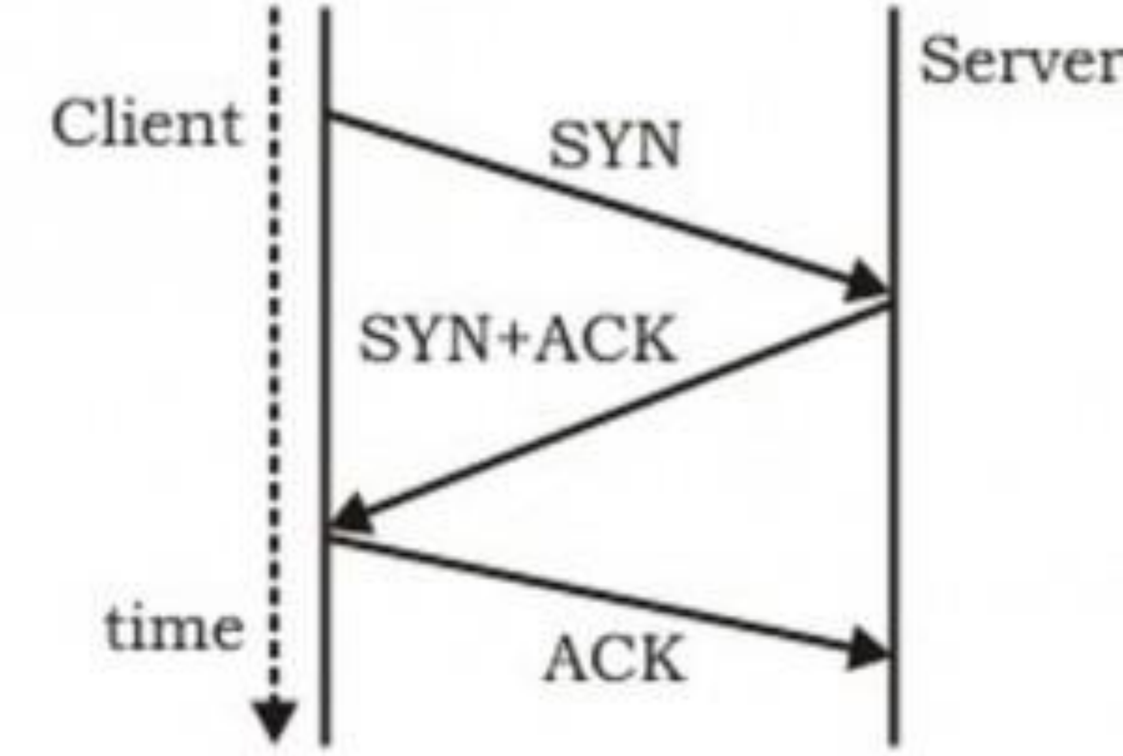
Q The three-way handshake for TCP connection establishment is shown below. Which of the following statements are TRUE?

(S_1) Loss of SYN + ACK from the server will not establish a connection

(S_2) Loss of ACK from the client cannot establish the connection

(S_3) The server moves LISTEN → SYN_RCVD → SYN_SENT → ESTABLISHED in the state machine on no packet loss

(S_4) The server moves LISTEN → SYN_RCVD → ESTABLISHED in the state machine on no packet loss. **(Gate-2008) (2 Marks)**



(A) S_2 and S_3 only

(B) S_1 and S_4

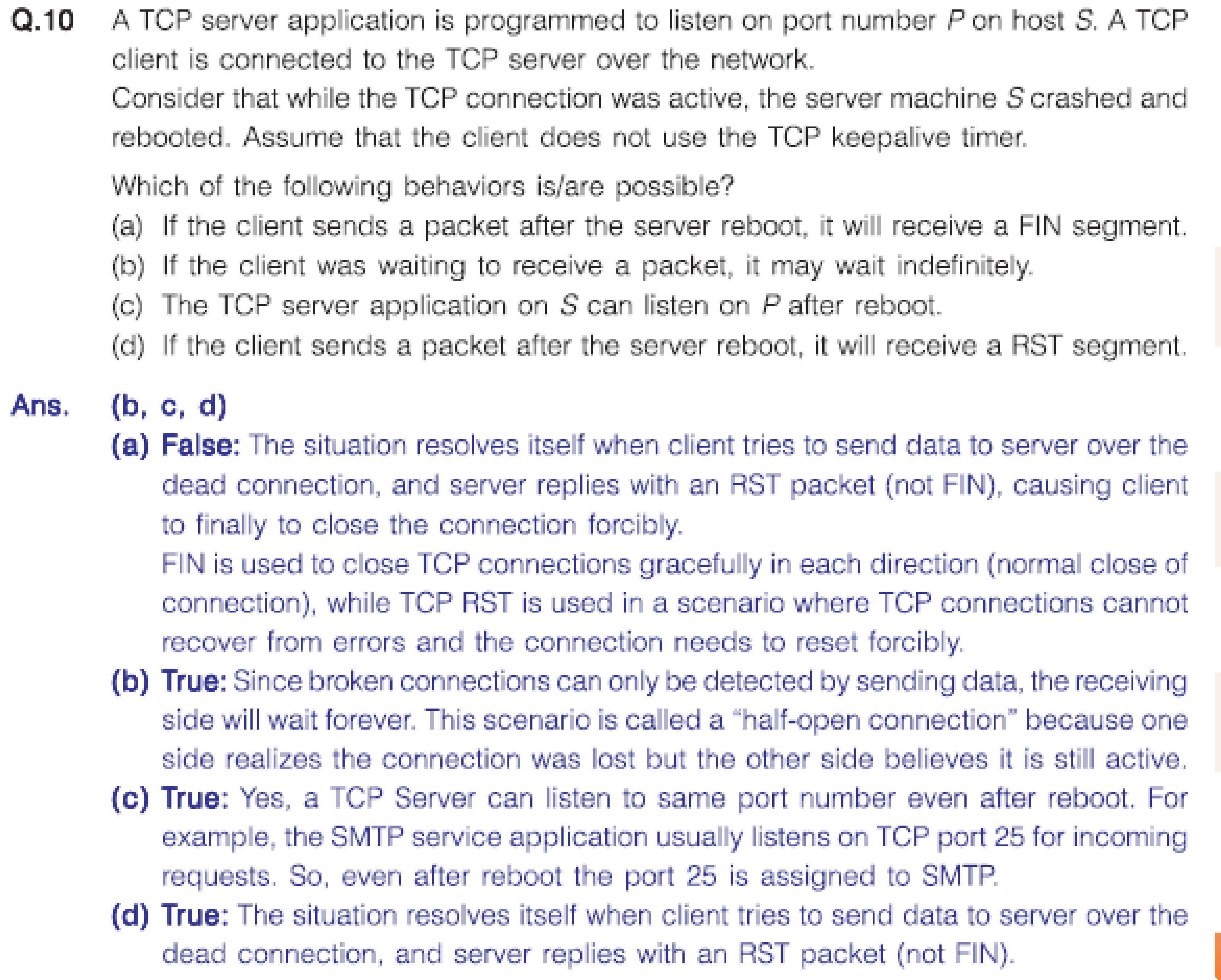
(C) S_1 and S_3

(D) S_2 and S_4

Q Consider a TCP connection in a state where there are no outstanding ACKs. The sender sends two segments back to back. The sequence numbers of the first and second segments are 230 and 290 respectively. The first segment was lost, but the second segment was received correctly by the receiver. Let X be the amount of data carried in the first segment (in bytes), and Y be the ACK number sent by the receiver. The values of X and Y (in that order) are **(Gate-2007) (1 Marks)**

(A) 60 and 290	**(B)** 230 and 291
(C) 60 and 231	**(D)** 60 and 230

(Gate-2021) (1 Marks)



Q.10 A TCP server application is programmed to listen on port number *P* on host *S*. A TCP client is connected to the TCP server over the network.

Consider that while the TCP connection was active, the server machine *S* crashed and rebooted. Assume that the client does not use the TCP keepalive timer.

Which of the following behaviors is/are possible?

(a) If the client sends a packet after the server reboot, it will receive a FIN segment.

(b) If the client was waiting to receive a packet, it may wait indefinitely.

(c) The TCP server application on *S* can listen on *P* after reboot.

(d) If the client sends a packet after the server reboot, it will receive a RST segment.

Ans. **(b, c, d)**

(a) False: The situation resolves itself when client tries to send data to server over the dead connection, and server replies with an RST packet (not FIN), causing client to finally to close the connection forcibly.

FIN is used to close TCP connections gracefully in each direction (normal close of connection), while TCP RST is used in a scenario where TCP connections cannot recover from errors and the connection needs to reset forcibly.

(b) True: Since broken connections can only be detected by sending data, the receiving side will wait forever. This scenario is called a "half-open connection" because one side realizes the connection was lost but the other side believes it is still active.

(c) True: Yes, a TCP Server can listen to same port number even after reboot. For example, the SMTP service application usually listens on TCP port 25 for incoming requests. So, even after reboot the port 25 is assigned to SMTP.

(d) True: The situation resolves itself when client tries to send data to server over the dead connection, and server replies with an RST packet (not FIN).