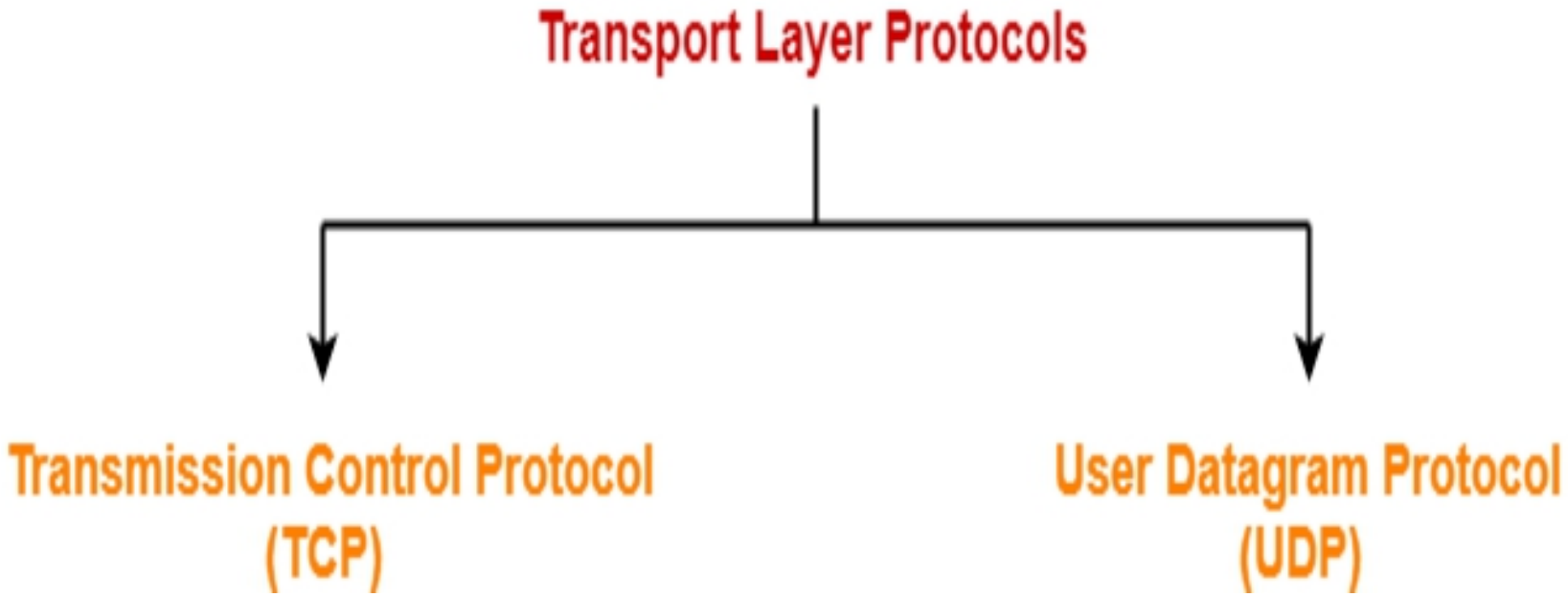
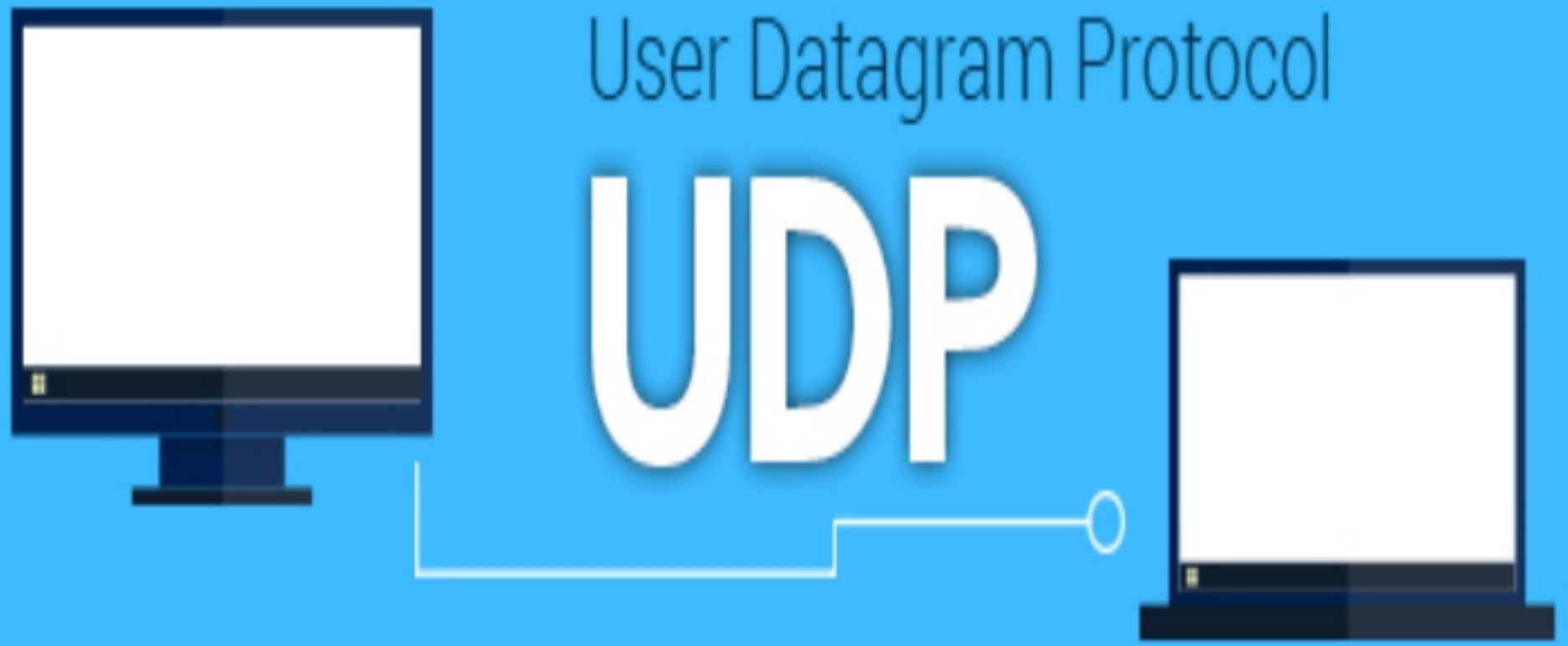
CN: WEEK 7

Transport Layer Protocols


Transport Layer Protocols
Transmission Control Protocol
(TCP)
User Datagram Protocol
(UDP)

User Datagram Protocol

UDP



- User Datagram Protocol (UDP) is a Transport Layer protocol.
- It is the simplest transport layer protocol.
- It has been designed to send data packets over the Internet.
- It simply takes the datagram from the network layer, attaches its header and sends it to the user.

Characteristics of UDP:

- ✓ It is a connectionless protocol.
- ✓ It is a stateless protocol.
- ✓ It is an unreliable protocol.
- ✓ It is a fast protocol.
- ✓ It offers the minimal transport service.
- ✓ It is almost a null protocol.
- ✓ It does not guarantee in order delivery.
- ✓ It does not provide congestion control mechanism.
- ✓ It is a good protocol for data flowing in one direction.

Need of UDP

Though Transmission Control Protocol (TCP) is the dominant transport layer protocol used with most of Internet services; provides assured delivery, reliability and much more but all these services cost us with additional overhead and latency.

Here, UDP comes into picture. For the realtime services like computer gaming, voice or video communication, live conferences; we need UDP.

Since high performance is needed, UDP permits packets to be dropped instead of processing delayed packets. There is no error checking in UDP, so it also save bandwidth.

User Datagram Protocol (UDP) is more efficient in terms of both latency and bandwidth.

UDP Header

Source Port: **Source Port is a 16 bit field and it identifies the port of the sending application.**

Destination Port: **Destination Port is a 16 bit field and It identifies the port of the receiving application.**

Length: **Length is a 16 bit field and it identifies the combined length of UDP Header and Encapsulated data.**

Length = Length of UDP Header + Length of encapsulated data

Checksum: **Checksum is a 16 bit field used for error control. It is calculated on UDP Header, encapsulated data and IP pseudo header.**

Applications Using UDP-

- *Applications which require one response for one request use UDP. Example- DNS.*
- *Routing Protocols like RIP and OSPF use UDP because they have very small amount of data to be transmitted.*
- *Trivial File Transfer Protocol (TFTP) uses UDP to send very small sized files.*
- *Broadcasting and multicasting applications use UDP.*
- *Streaming applications like multimedia, video conferencing etc use UDP since they require speed over reliability.*
- *Real time applications like chatting and online games use UDP.*
- *Management protocols like SNMP (Simple Network Management Protocol) use UDP.*
- *Bootp / DHCP uses UDP.*
- *Other protocols that use UDP are- Kerberos, Network Time Protocol (NTP), Network News Protocol (NNP) etc.*

POINT 1: Size of UDP Header= 8 bytes

- Unlike TCP header, the size of UDP header is fixed.
- This is because in UDP header, all the fields are of definite size.
- Size of UDP Header = Sum of the size of all the fields = 8 bytes.

POINT 2: UDP is almost a null protocol.

- UDP provides very limited services.
- The only services it provides are checksumming of data and multiplexing by port number.

POINT 3: UDP is an unreliable protocol.

- UDP does not guarantee the delivery of datagram to its respective user (application).
- The lost datagrams are not retransmitted by UDP.

POINT 4: Checksum calculation is not mandatory in UDP.

- UDP is already an unreliable protocol and error checking does not make much sense.
- Also, time is saved and transmission becomes faster by avoiding to calculate it.

POINT 5: UDP does not guarantee in order delivery.

- UDP allows out of order delivery to ensure better performance.
- If some data is lost on the way, it does not call for retransmission and keeps transmitting data.

POINT 6: Application layer can perform some tasks through UDP.

- Application layer can do many tasks through UDP like Trace Route, Record Route and Time stamp.
- UDP acts like a messenger between the application layer and the IP datagram.


TCP

TCP is short for Transmission Control Protocol.

It is a transport layer protocol.

It has been designed to send data packets over the Internet.

It establishes a reliable end to end connection before sending any data.

Point 1: TCP is a reliable protocol.

- *It guarantees the delivery of data packets to its correct destination.*
- *After receiving the data packet, receiver sends an acknowledgement to the sender.*
- *It tells the sender whether data packet has reached its destination safely or not.*
- *TCP employs retransmission to compensate for packet loss.*

Point 2: TCP is a connection oriented protocol.

- ***TCP establishes an end to end connection between the source and destination.***
- ***The connection is established before exchanging the data.***
- ***The connection is maintained until the application programs at each end finishes exchanging the data.***

Point 3: TCP handles both congestion and flow control.

- ***TCP handles congestion and flow control by controlling the window size.***
- ***TCP reacts to congestion by reducing the sender window size.***

Point 4: TCP ensures in-order delivery.

- ***TCP ensures that the data packets get deliver to the destination in the same order they are sent by the sender.***
- ***Sequence Numbers are used to coordinate which data has been transmitted and received.***

Point 5: TCP connections are full duplex.

- ***TCP connection allows to send data in both the directions at the same time.***
- ***So, TCP connections are Full Duplex.***

Point 6: TCP works in collaboration with Internet Protocol.

- ***A TCP connection is uniquely identified by using-***
  Combination of port numbers and IP Addresses of sender and receiver.
- ***IP Addresses indicate which systems are communicating.***
- ***Port numbers indicate which end to end sockets are communicating.***
- ***Port numbers are contained in the TCP header and IP Addresses are contained in the IP header.***
- ***TCP segments are encapsulated into an IP datagram.***
- ***So, TCP header immediately follows the IP header during transmission.***

Point 7: TCP can use both selective & cumulative acknowledgements.

- *TCP uses a combination of Selective Repeat and Go back N protocols.*
- *In TCP, sender window size = receiver window size.*
- *In TCP, out of order packets are accepted by the receiver.*
- *When receiver receives an out of order packet, it accepts that packet but sends an acknowledgement for the expected packet.*
- *Receiver may choose to send independent acknowledgements or cumulative acknowledgement.*
- *To sum up, TCP is a combination of 75% SR protocol and 25% Go back N protocol.*

Point 8: TCP is a byte stream protocol.

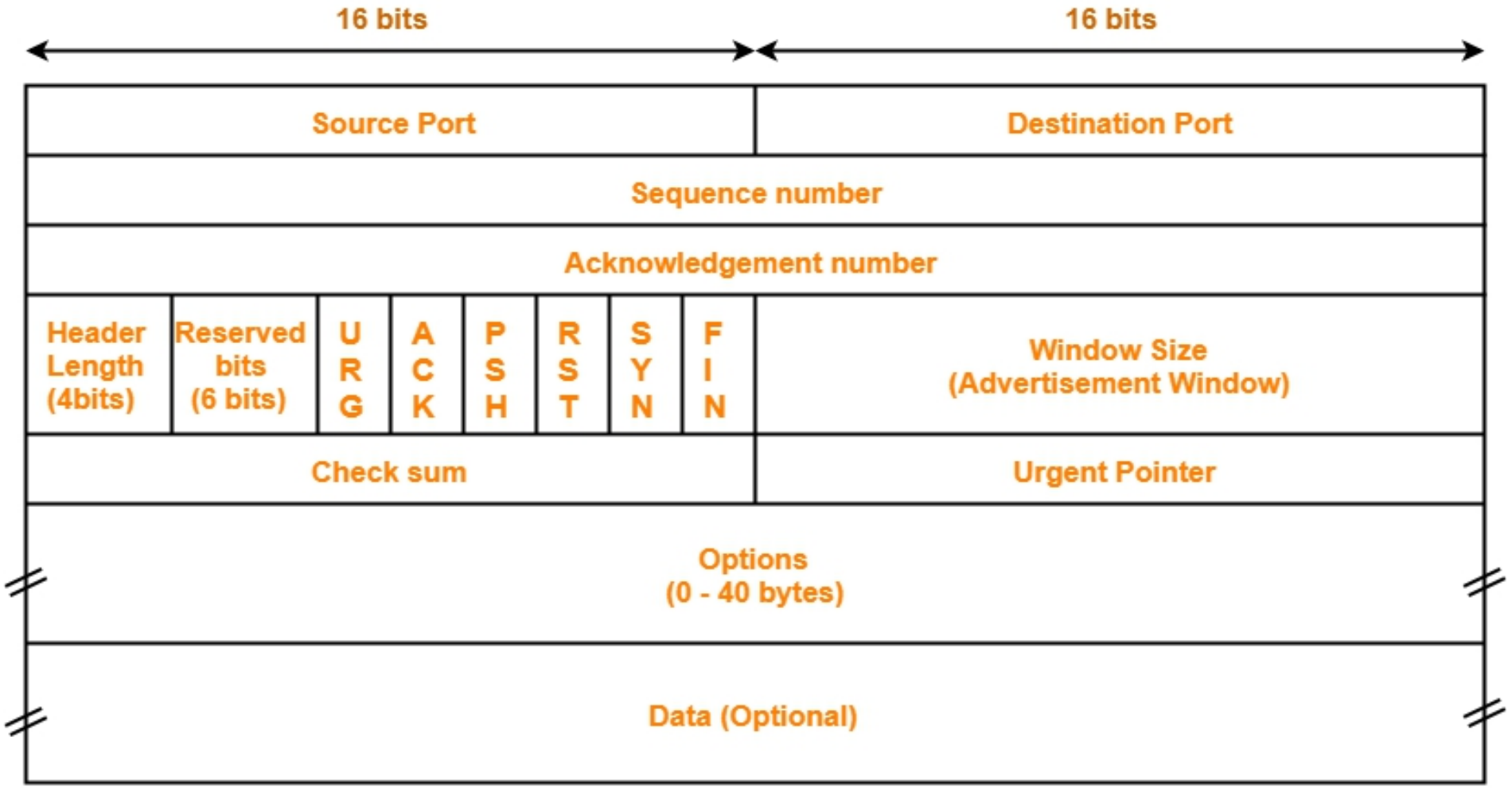
- *Application layer sends data to the transport layer without any limitation.*
- *TCP divides the data into chunks where each chunk is a collection of bytes.*
- *Then, it creates a TCP segment by adding IP header to the data chunk.*
- *TCP segment = TCP header + Data chunk.*

Point 9: TCP provides error checking & recovery mechanism.

- ***TCP provides error checking and recovery using three simple techniques-***

- ✓ ***Checksum***
- ✓ ***Acknowledgement***
- ✓ ***Retransmission***

TCP Header



TCP Header

Source Port: It is a 16 bit field that identifies the port of the sending application.

Destination Port: It is a 16 bit field that identifies the port of the receiving application.

NOTE

It is important to note-

- A TCP connection is uniquely identified by using-
  Combination of port numbers and IP Addresses of sender and receiver
- IP Addresses indicate which systems are communicating.
- Port numbers indicate which end to end sockets are communicating.

Sequence Number: It is a 32 bit field. TCP assigns a unique sequence number to each byte of data contained in the TCP segment. This field contains the sequence number of the first data byte.

Acknowledgement Number: It is a 32 bit field that contains sequence number of the data byte that receiver expects to receive next from the sender. It is always sequence number of the last received data byte incremented by 1.

Header Length: It is a 4 bit field that contains the length of TCP header. It helps in knowing from where the actual data begins. The length of TCP header always lies in the range-

[20 bytes , 60 bytes]

Reserved Bits: The 6 bits are reserved and are not used.

URG Bit: When URG bit is set to 1,

✓ It indicates the receiver that certain amount of data within the current segment is urgent.

✓ Urgent data is pointed out by evaluating the urgent pointer field.

✓ The urgent data has be prioritized.

✓ Receiver forwards urgent data to the receiving application on a separate channel.

ACK Bit: ACK bit indicates whether acknowledgement number field is valid or not. When ACK bit is set to 1, it indicates that acknowledgement number contained in the TCP header is valid.

PSH Bit: PSH bit is used to push the entire buffer immediately to the receiving application. When PSH bit is set to 1, all the segments in the buffer are immediately pushed to the receiving application.

RST Bit: RST bit is used to reset the TCP connection. When RST bit is set to 1, it indicates the receiver to terminate the connection immediately. It causes both the sides to release the connection and all its resources abnormally. This is used only when there are unrecoverable errors.

SYN Bit: SYN bit is used to synchronize the sequence numbers. When SYN bit is set to 1, it indicates the receiver that the sequence number contained in the TCP header is the initial sequence number.

FIN Bit: FIN bit is used to terminate the TCP connection. When FIN bit is set to 1, it indicates the receiver that the sender wants to terminate the connection.

Window Size: It is a 16 bit field that contains the size of the receiving window of the sender. It advertises how much data (in bytes) the sender can receive without acknowledgement. Thus, window size is used for Flow Control.

Checksum: It is a 16 bit field used for error control. It verifies the integrity of data in the TCP payload. Sender adds CRC checksum to the checksum field before sending the data. Receiver rejects the data that fails the CRC check.

Urgent Pointer: It is a 16 bit field that indicates how much data in the current segment counting from the first data byte is urgent. Urgent pointer added to the sequence number indicates the end of urgent data byte.

Options: It is used for several purposes. The size of options field vary from 0 bytes to 40 bytes. Options field is generally used for the following purposes like Time stamp, Window size extension. Parameter negotiation and Padding.

TCP Sequence Number

- *Each TCP segment sent by the sender contains some bytes of data.*
- *TCP assigns a unique number to each data byte for its identification.*
- *This unique number is called as TCP Sequence Number.*
- *In TCP header, sequence number is a 32 bit field.*
- *So, maximum number of possible sequence numbers = 2^{32}.*
- *These sequence numbers lie in the range [0 , $2^{32} - 1$].*

Sequence number serves the following purposes-

- ✓ **It helps to identify each data byte uniquely.**
- ✓ **It helps in the segmentation of data into TCP segments and reassemble them later.**
- ✓ **It helps to keep track of how much data has been transferred and received.**
- ✓ **It helps to put the data back into the correct order if it is received in the wrong order.**
- ✓ **It helps to request data when it has been lost in transit.**

3 Way Handshake (TCP Connection)

Three Way Handshake is a process used for establishing a TCP connection.

Consider-

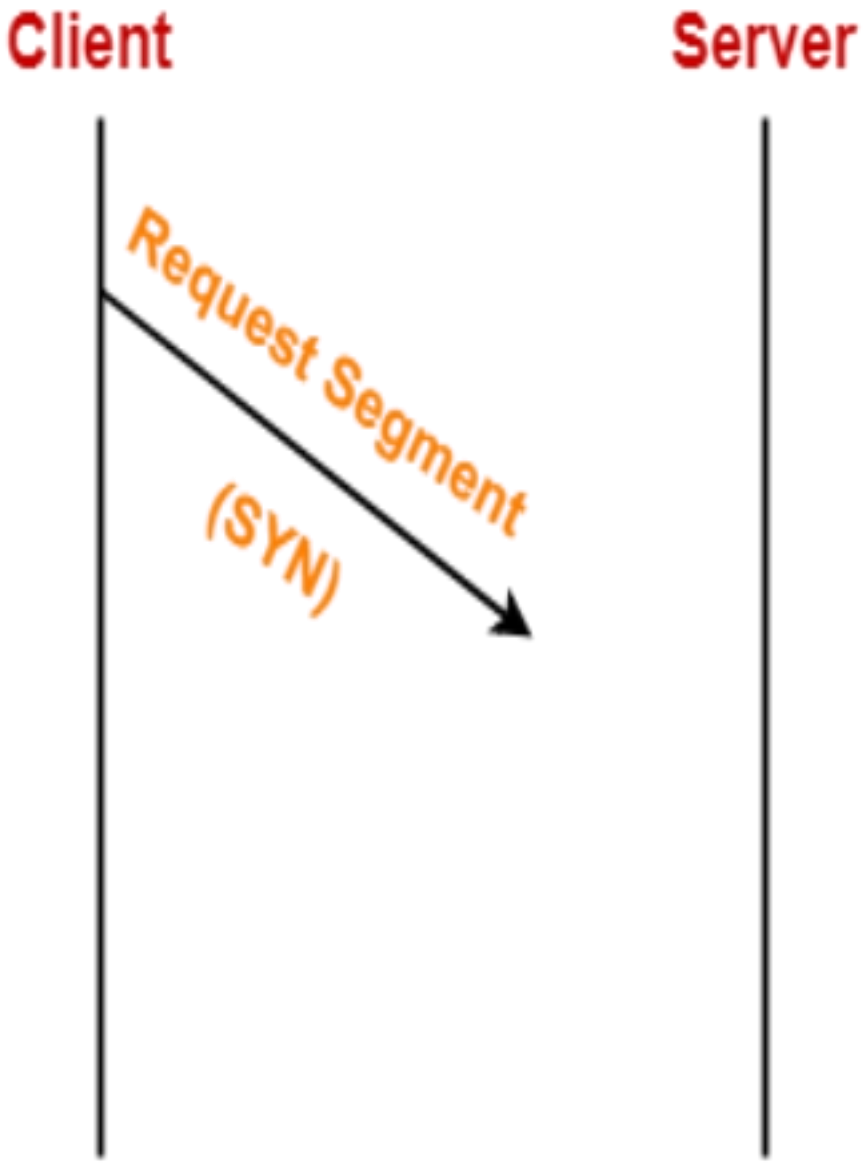
Client wants to establish a connection with the server.
Before Three Way Handshake, both client and server are in closed state.

TCP Handshake involves the following steps in establishing the connection:::

Step-1: SYN-

For establishing a connection,

- Client sends a request segment to the server.
- Request segment consists only of TCP Header with an empty payload.
- Then, it waits for a reply segment from the server.



Request segment contains the following information in TCP header-

- ✓ ***Initial sequence number***
- ✓ ***SYN bit set to 1***
- ✓ ***Maximum segment size***
- ✓ ***Receiving window size***

Initial Sequence Number:

Client sends the initial sequence number to the server.
It is contained in the sequence number field.
It is a randomly chosen 32 bit value.

SYN Bit Set To 1:

Client sets SYN bit to 1 which indicates the server.
This segment contains the initial sequence number used by the client.
It has been sent for synchronizing the sequence numbers.

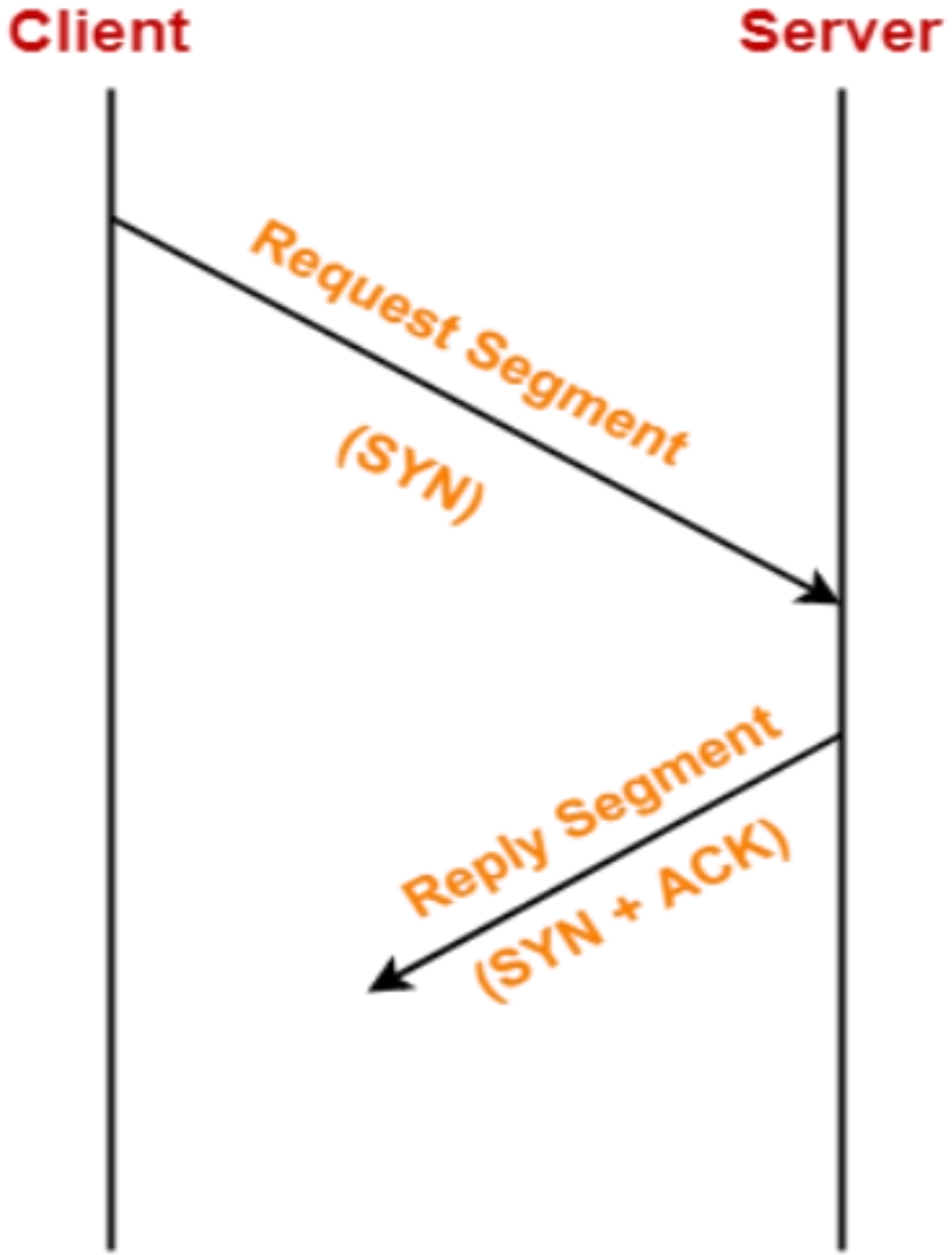
Maximum Segment Size (MSS):

Client sends its MSS to the server.
It dictates the size of the largest data chunk that client can send and receive from the server.

Receiving Window Size:

Client sends its receiving window size to the server.
It dictates the limit of unacknowledged data that can be sent to the client.
It is contained in the window size field.

Step-2: SYN + ACK-



After receiving the request segment,

- **Server responds to the client by sending the reply segment.**
- **It informs the client of the parameters at the server side.**

Reply segment contains the following information in TCP header-

- ✓ ***Initial sequence number***
- ✓ ***SYN bit set to 1***
- ✓ ***Maximum segment size***
- ✓ ***Receiving window size***
- ✓ ***Acknowledgment number***
- ✓ ***ACK bit set to 1***

Initial Sequence Number:

Server sends the initial sequence number to the client.
It is contained in the sequence number field.
It is a randomly chosen 32 bit value.

SYN Bit Set To 1:

Server sets SYN bit to 1 which indicates the client-
This segment contains the initial sequence number used by the server.
It has been sent for synchronizing the sequence numbers.

Maximum Segment Size (MSS):

Server sends its MSS to the client.
It dictates the size of the largest data chunk that server can send and receive from the client.

Receiving Window Size:

Server sends its receiving window size to the client.
It dictates the limit of unacknowledged data that can be sent to the server.
It is contained in the window size field.

Acknowledgement Number:

Server sends the initial sequence number incremented by 1 as an acknowledgement number.
It dictates the sequence number of the next data byte that server expects to receive from the client.

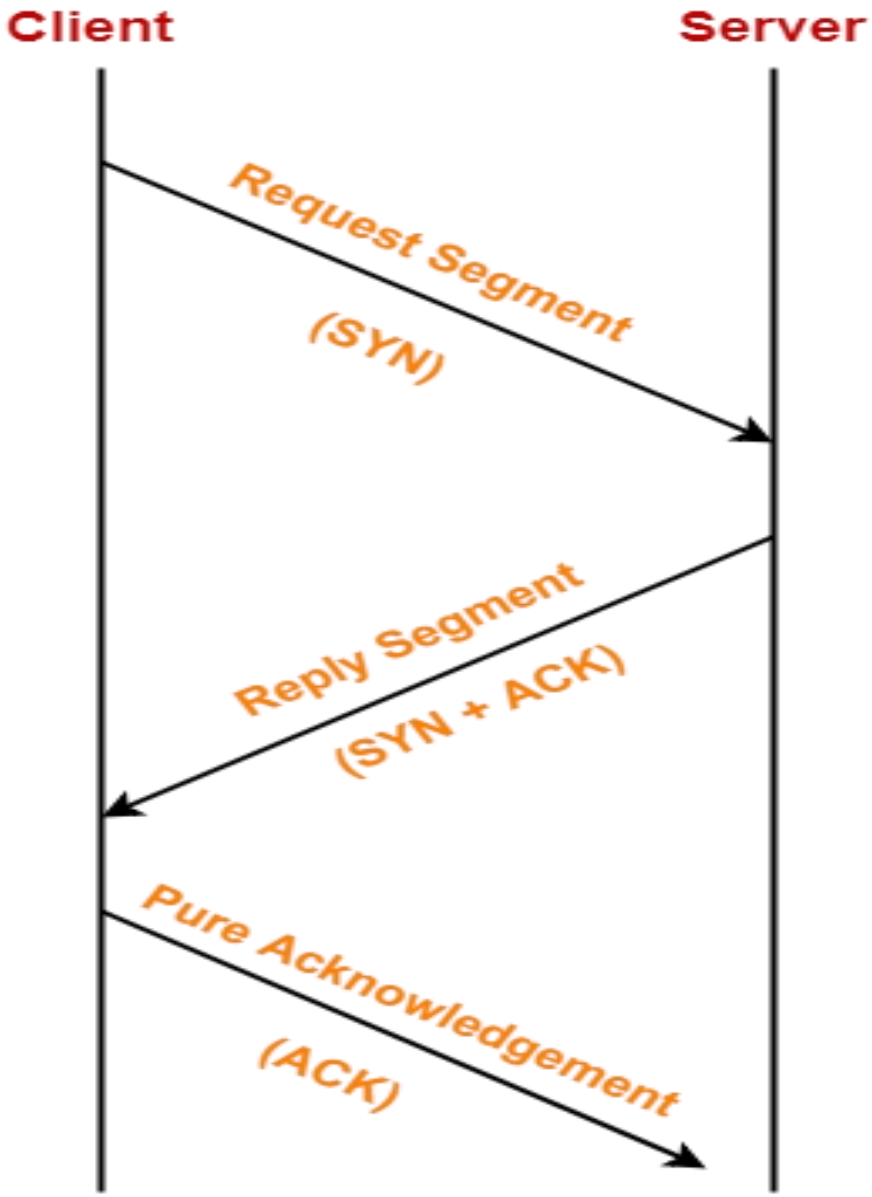
ACK Bit Set To 1:

Server sets ACK bit to 1.
It indicates the client that the acknowledgement number field in the current segment is valid.

Step-3: ACK-

After receiving the reply segment,

- Client acknowledges the response of server.
- It acknowledges the server by sending a pure acknowledgement.



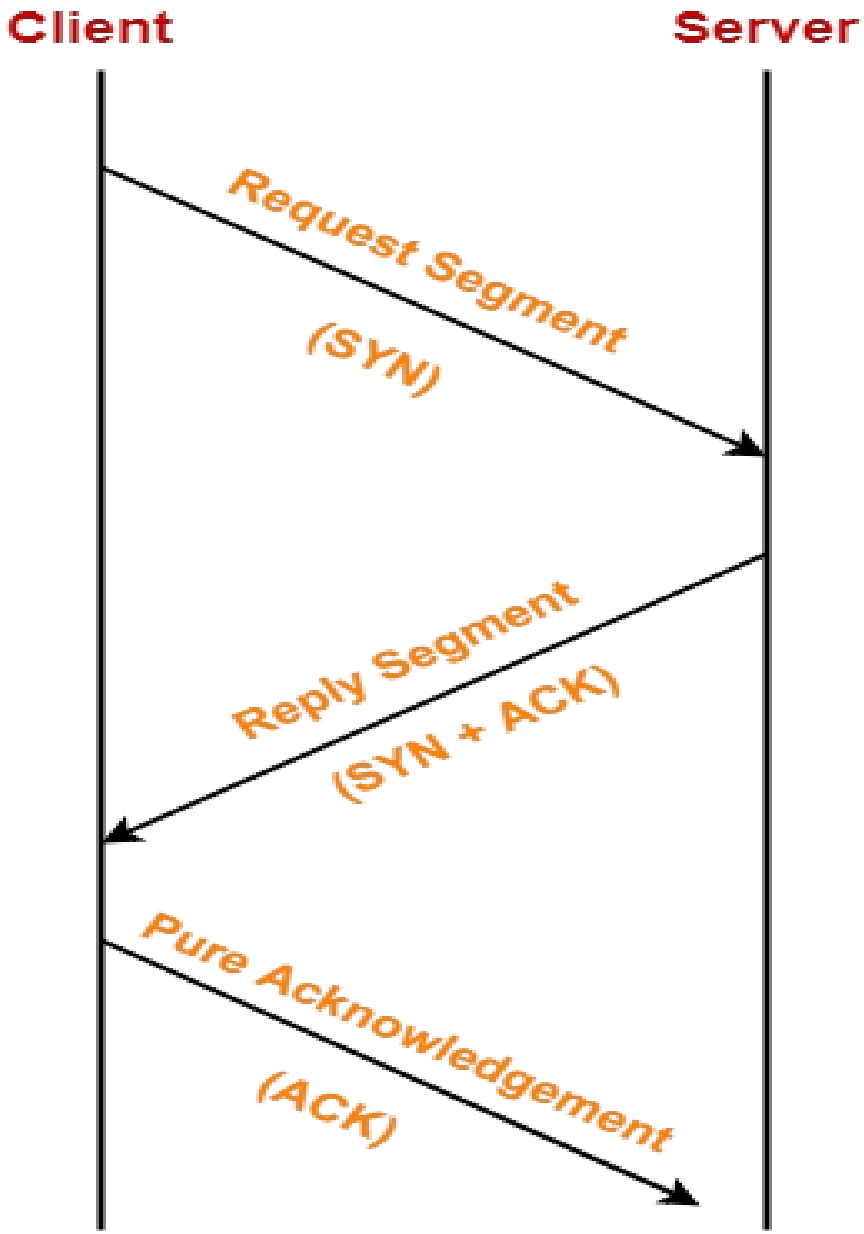
3 Way Handshake

With these, a Full Duplex connection is established.

Thus we have discussed-

TCP uses Three Way Handshake to establish a connection between the sender and receiver.

Connection establishment using Three Way Handshake involves the steps as shown-



3 Way Handshake

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.**

When TCP Retransmission Occurs?

When sender discovers that the segment sent by it is lost, it retransmits the same segment to the receiver.

Sender discovers that the TCP segment is lost when-

- ✓ **Either Time Out Timer expires**
- ✓ **Or it receives three duplicate acknowledgements**

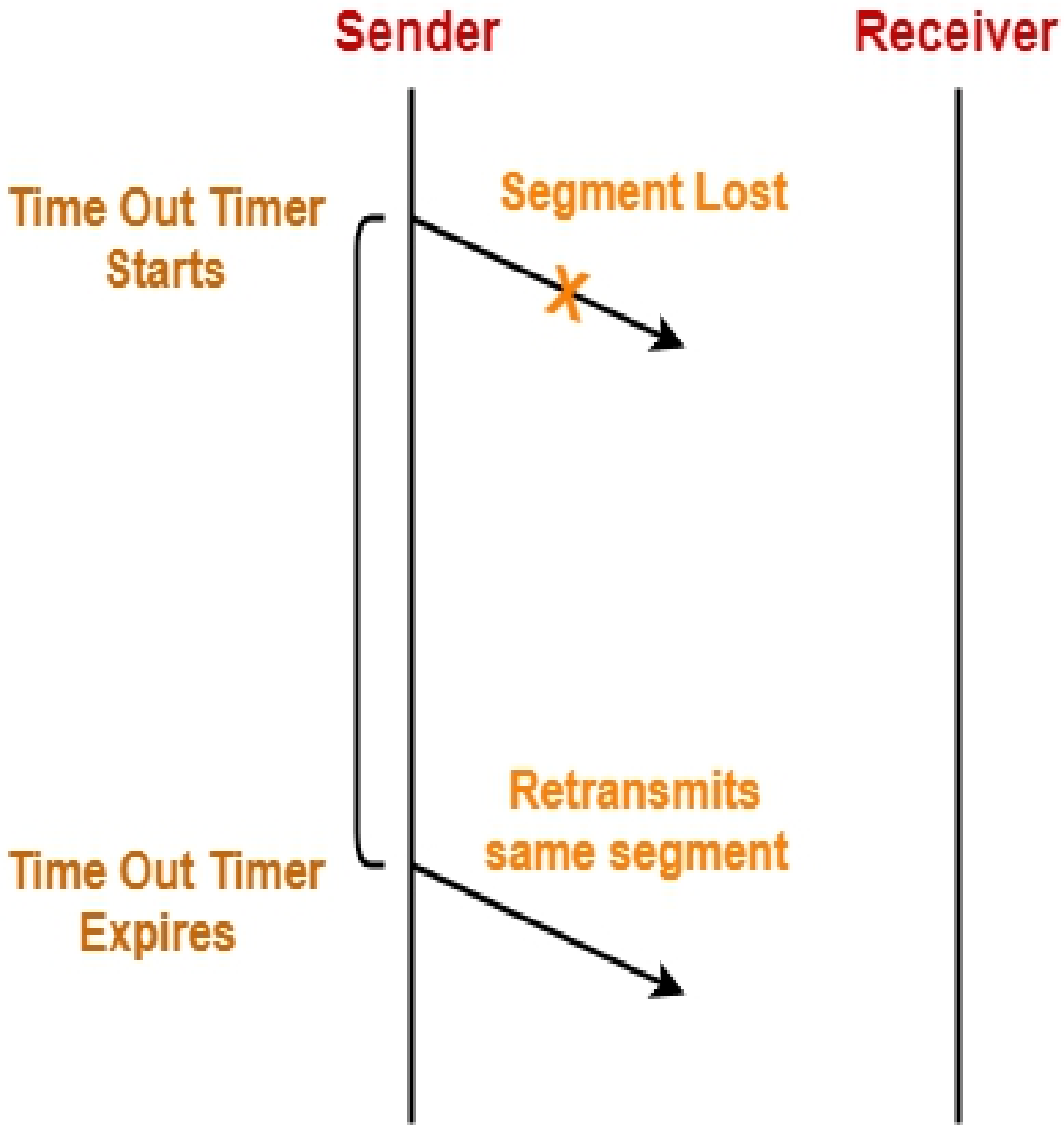
Retransmission After Time Out Timer Expiry

Each time sender transmits a TCP segment to the receiver, it starts a Time Out Timer.

Now, following two cases are possible-

Case-01:
Sender receives an acknowledgement for the sent segment before the timer goes off. In this case, sender stops the timer.

Case-02:
Sender does not receives 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.



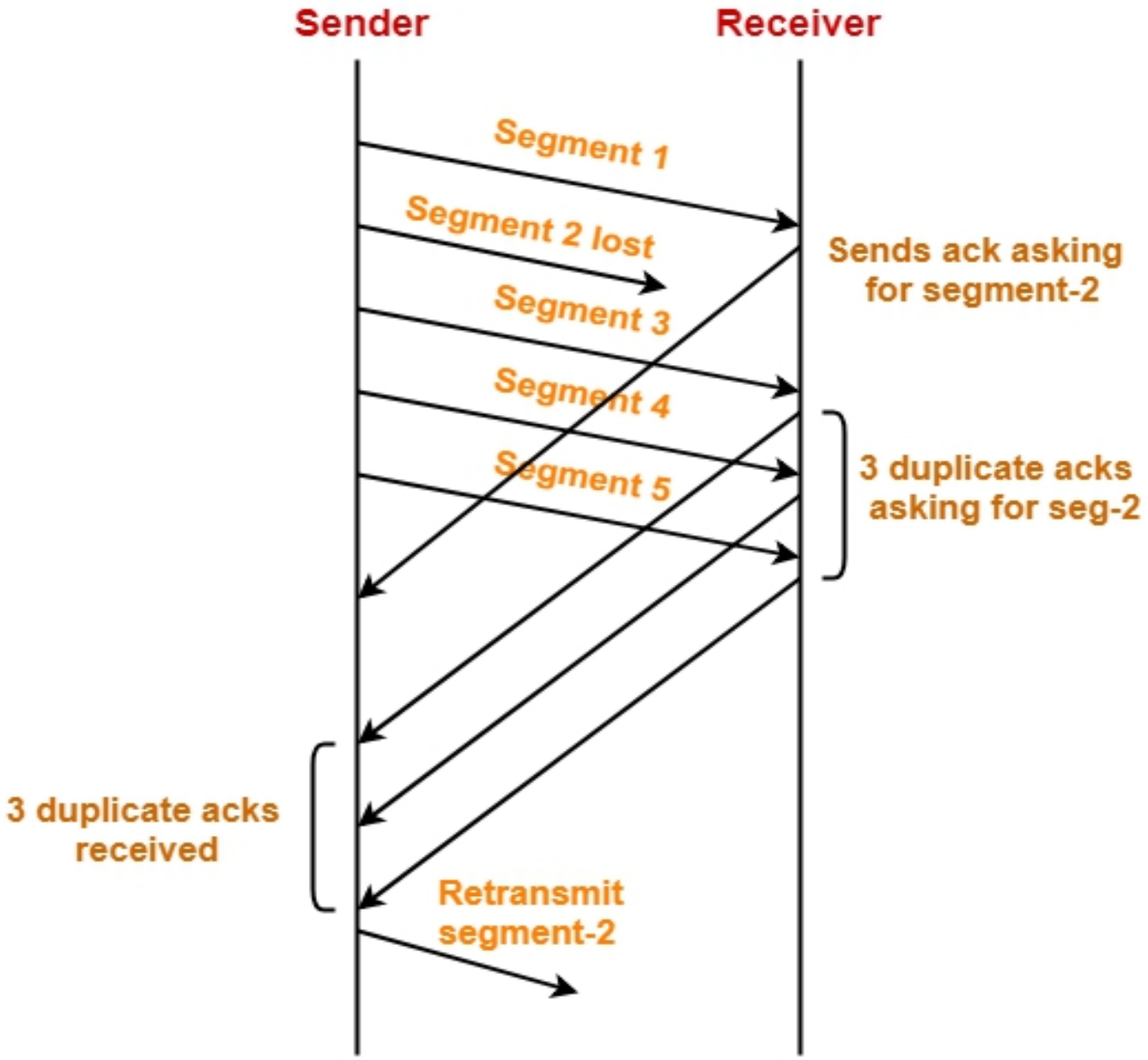
Retransmission after Time Out Timer Expiry

Retransmission After Receiving 3 Duplicate Acknowledgements-

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.

As shown in figure, sender sends 5 TCP segments to the receiver. The second TCP segment gets lost before reaching the receiver. 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.



Retransmission after receiving 3 duplicate acks

Important Points

Consider time out timer expires before receiving the acknowledgement for a TCP segment.
This case suggests the stronger possibility of congestion in the network.

Consider sender receives 3 duplicate acknowledgements for the same TCP segment.
This case suggests the weaker possibility of congestion in the network.

Consider receiver does not receives 3 duplicate acknowledgements for the lost TCP segment.
In such a case, retransmission occurs only after time out timer goes off.

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

TCP Connection Termination

A TCP connection is terminated using FIN segment where FIN bit is set to 1.

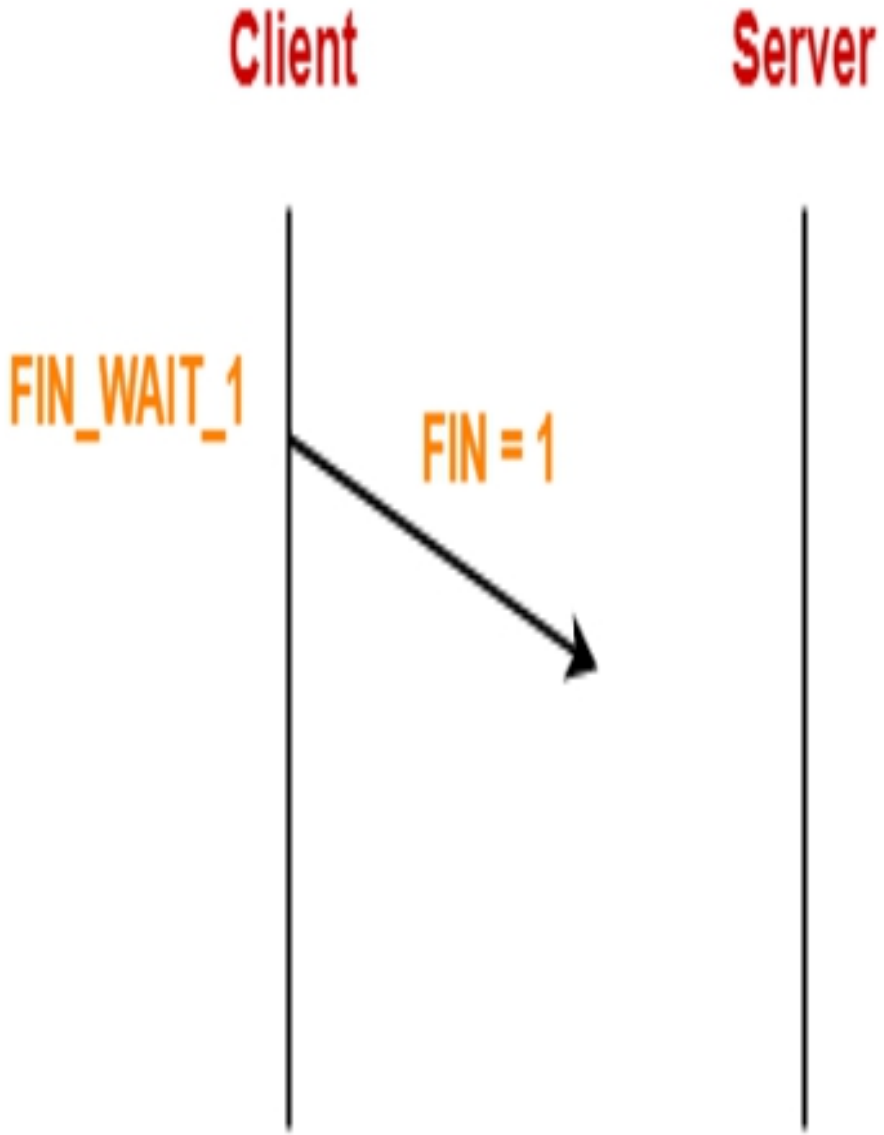
Consider-

- **There is a well established TCP connection between the client and server.**
- **Client wants to terminate the connection.**
- **The following steps are followed in terminating the connection-**

Step-01:

For terminating the connection,

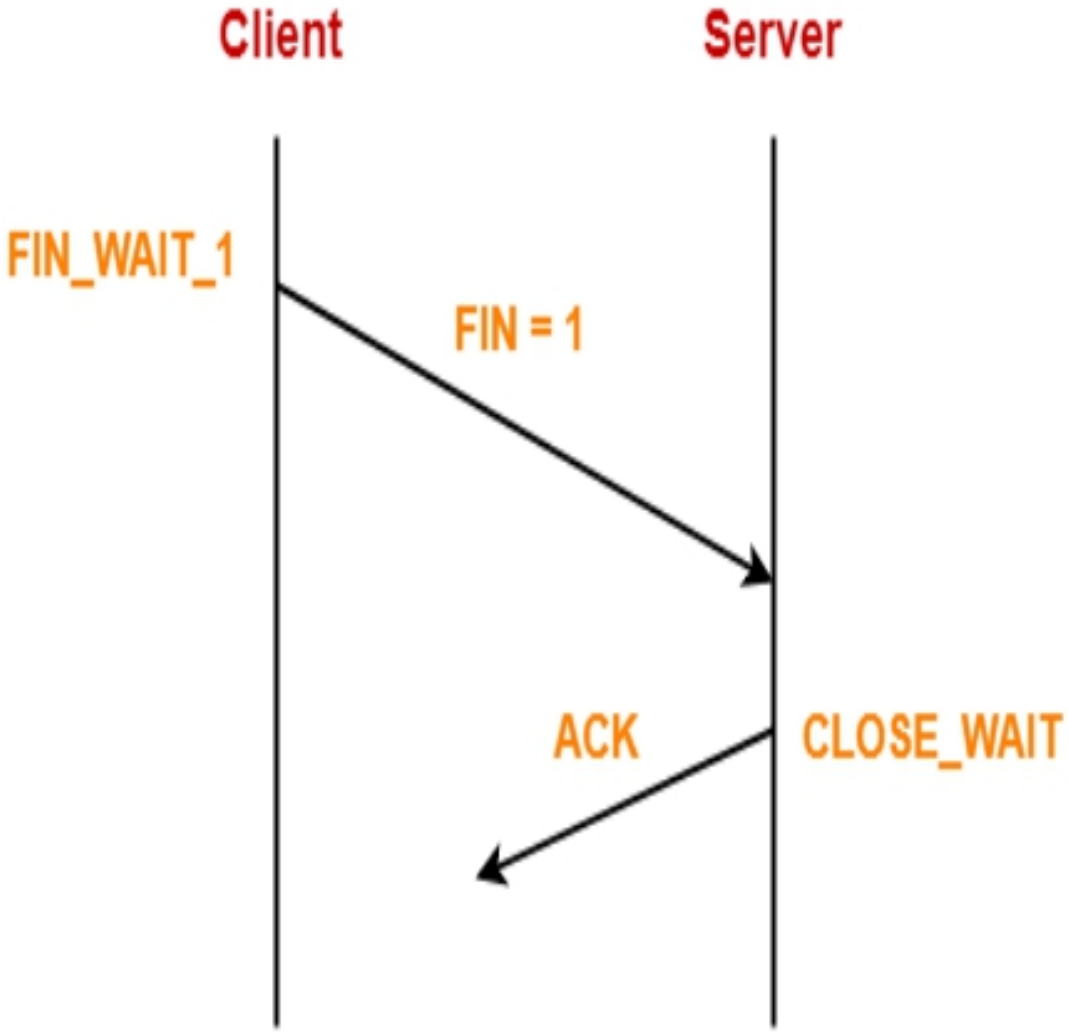
- Client sends a FIN segment to the server with FIN bit set to 1.
- Client enters the FIN_WAIT_1 state.
- Client waits for an acknowledgement from the server.



Step-02:

After receiving the FIN segment,

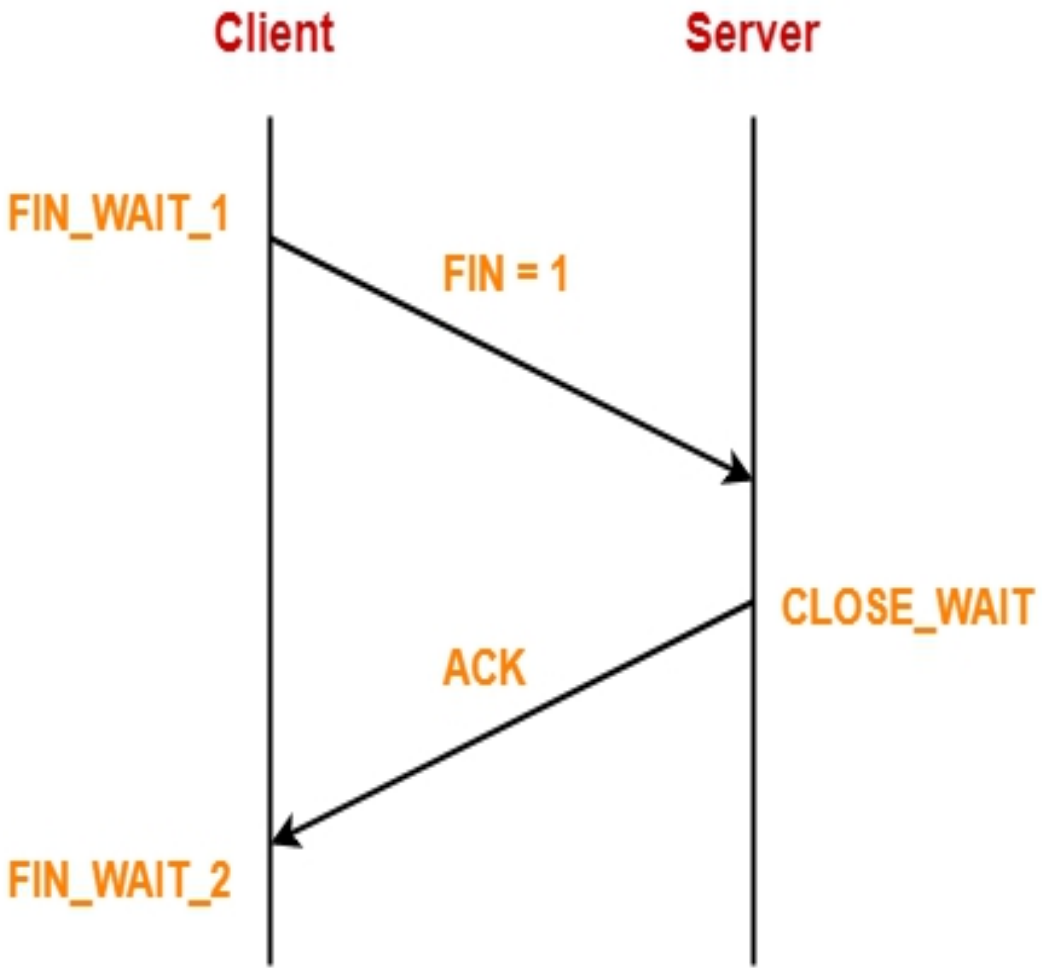
- Server frees up its buffers.
- Server sends an acknowledgement to the client.
- Server enters the CLOSE_WAIT state.



Step-03:

After receiving the acknowledgement, client enters the FIN_WAIT_2 state.

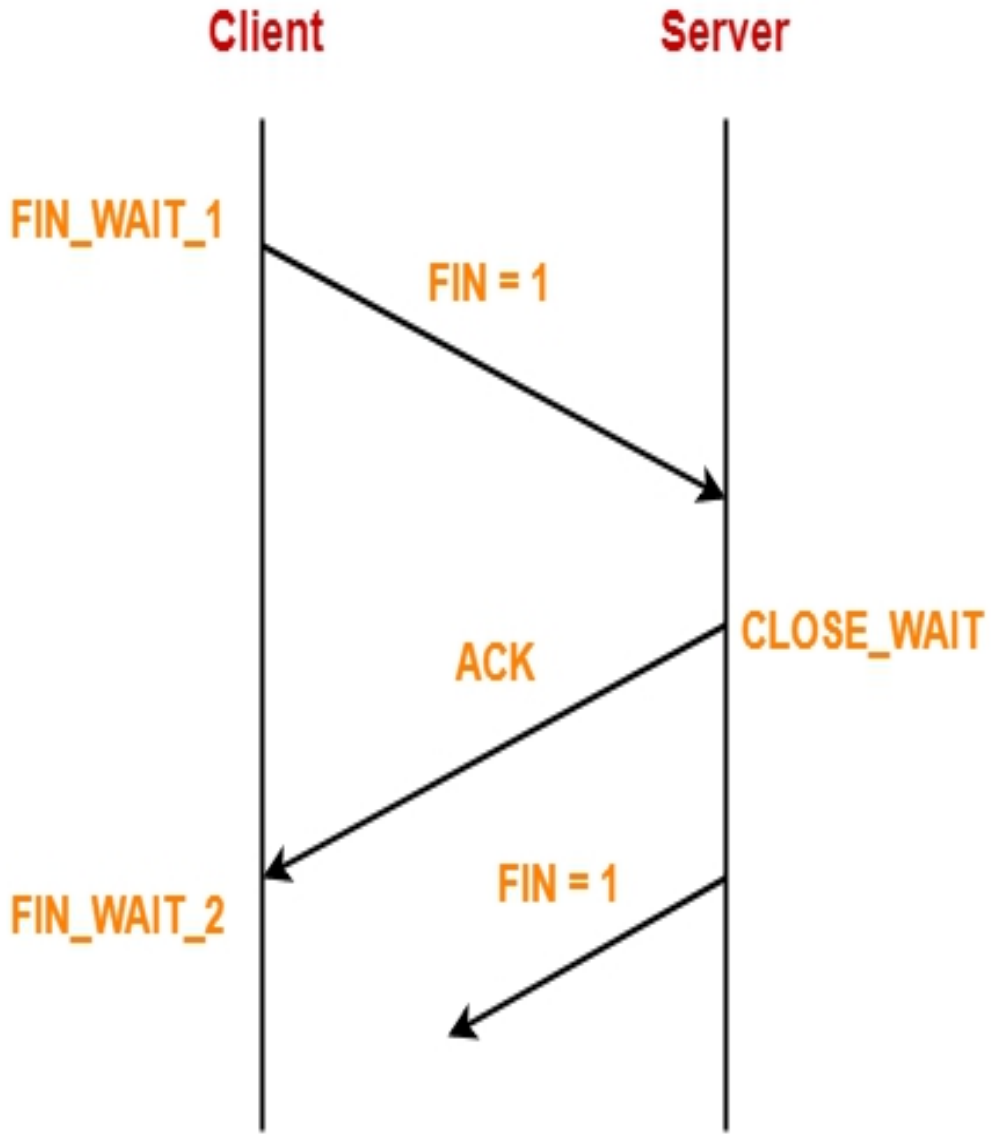
- The connection from client to server is terminated i.e. one way connection is closed.
- Client can not send any data to the server since server has released its buffers.
- Pure acknowledgements can still be sent from the client to server.
- The connection from server to client is still open i.e. one way connection is still open.
- Server can send both data and acknowledgements to the client.



Step-04:

Now, suppose server wants to close the connection with the client. For terminating the connection,

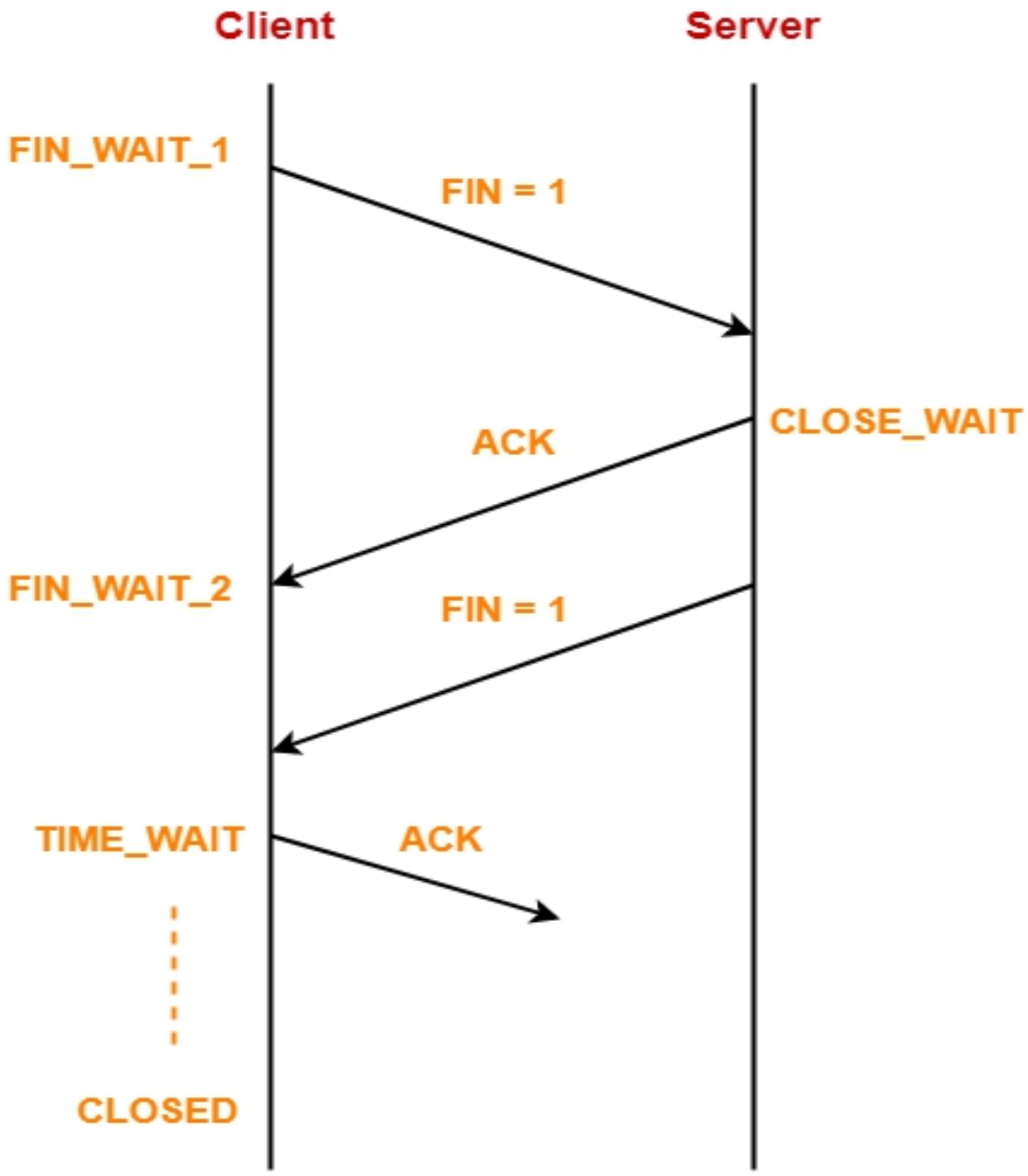
- Server sends a FIN segment to the client with FIN bit set to 1.
- Server waits for an acknowledgement from the client.



Step-05:

After receiving the FIN segment,

- Client frees up its buffers.
- Client sends an acknowledgement to the server.
- Client enters the TIME_WAIT state.



TIME_WAIT State-

- The TIME_WAIT state allows the client to resend the final acknowledgement if it gets lost.
- The time spent by the client in TIME_WAIT state depends on the implementation.
- The typical values are 30 seconds, 1 minute and 2 minutes.
- After the wait, the connection gets formally closed.

Q. If WAN link is 2 Mbps and RTT between source and destination is 300 msec, what would be the optimal TCP window size needed to fully utilize the line?

Given-

- Bandwidth = 2 Mbps
- RTT = 300 msec

Optimal TCP window size

= Maximum amount of data that can be sent in 1 RTT

= 2 Mbps x 300 msec

= 600 x 10^3 bits

= 60,0000 bits

= 75,000 bytes

Q. Suppose host A is sending a large file to host B over a TCP connection. The two end hosts are 10 msec apart (20 msec RTT) connected by a 1 Gbps link. Assume that they are using a packet size of 1000 bytes to transmit the file. For simplicity, ignore ack packets. At least how big would the window size (in packets) have to be for the channel utilization to be greater than 80%?

Given-

- RTT = 20 msec
- Bandwidth = 1 Gbps
- Packet size = 1000 bytes
- Efficiency >= 80%

Window Size For 100% Efficiency-

For 100% efficiency,

Window size

= Maximum number of bits that can be transmitted in 1 RTT

= 1 Gbps x 20 msec

= (10^9 bits per sec) x 20 x 10^{-3} sec

= 20 x 10^6 bits = 2 x 10^7 bits

Window Size For 80% Efficiency-

For 80% efficiency,

Window size

= 0.8 x 2 x 10^7 bits

= 1.6 x 10^7 bits

In terms of packets,

Window size

= 1.6 x 10^7 bits / Packet size

= 1.6 x 10^7 bits / (1000 x 8 bits)

= 0.2 x 10^4 packets

= 2000 packets

Q. A TCP machine is sending windows of 65535 B over a 1 Gbps channel that has a 10 msec one way delay.
What is the maximum throughput achievable?
What is the line efficiency?

Given-

- Window size = 65535 bytes
- Bandwidth = 1 Gbps
- One way delay = 10 msec

Maximum amount of data that can be sent in 1 RTT

= 1 Gbps x (2 x 10 msec)

= (10^9 bits per sec) x 20 x 10^{-3} sec

= 20 x 10^6 bits = 25 x 10^5 bytes

Amount of data that is actually being sent in 1 RTT = 65535 bytes

Thus,

Line Efficiency(η)

= Amount of data being sent in 1 RTT / Maximum amount of data that can be sent in 1 RTT

= 65535 bytes / 25 x 10^5 bytes = 0.026214 = 2.62%

Now,

Maximum Achievable Throughput

= Efficiency x Bandwidth

= 0.0262 x 1 Gbps = 26.214 Mbps

TCP Congestion Control

Congestion refers to a network state where the message traffic becomes so heavy that it slows down the network response time.

- **Congestion is an important issue that can arise in Packet Switched Network.**
- **Congestion leads to the loss of packets in transit.**
- **So, it is necessary to control the congestion in network.**
- **It is not possible to completely avoid the congestion.**

Now, let us discuss how congestion is handled at TCP.

Congestion control refers to techniques and mechanisms that can either prevent congestion before it happens or remove congestion after it has happened

TCP Congestion Control

TCP reacts to congestion by reducing the sender window size.

The size of the sender window is determined by the following two factors:

- ✓ Receiver window size
- ✓ Congestion window size

Sender window size =

Min (Receiver window size, Congestion window size)

Receiver Window Size-

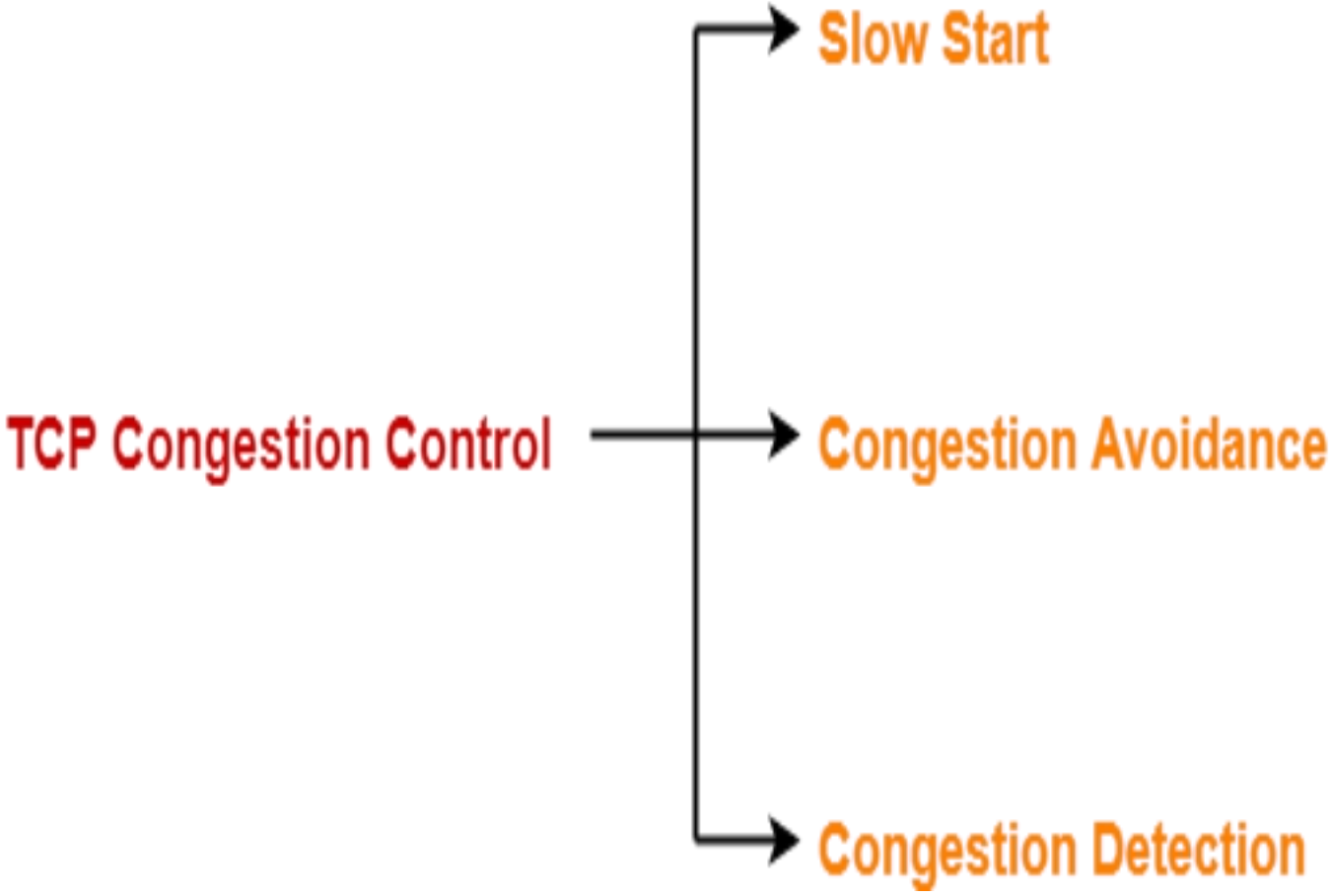
- Receiver window size is an advertisement of "How much data (in bytes) the receiver can receive without acknowledgement?"
- Sender should not send data greater than receiver window size.
- Otherwise, it leads to dropping the TCP segments which causes TCP Retransmission.
- So, sender should always send data less than or equal to receiver window size.
- Receiver dictates its window size to the sender through TCP Header.

Congestion Window Size-

- Sender should not send data greater than congestion window size.
- Otherwise, it leads to dropping the TCP segments which causes TCP Retransmission.
- So, sender should always send data less than or equal to congestion window size.
- Different variants of TCP use different approaches to calculate the size of congestion window.
- Congestion window is known only to the sender and is not sent over the links.

TCP Congestion Policy

TCP's general policy for handling congestion consists of following three phases-



Slow Start Phase-

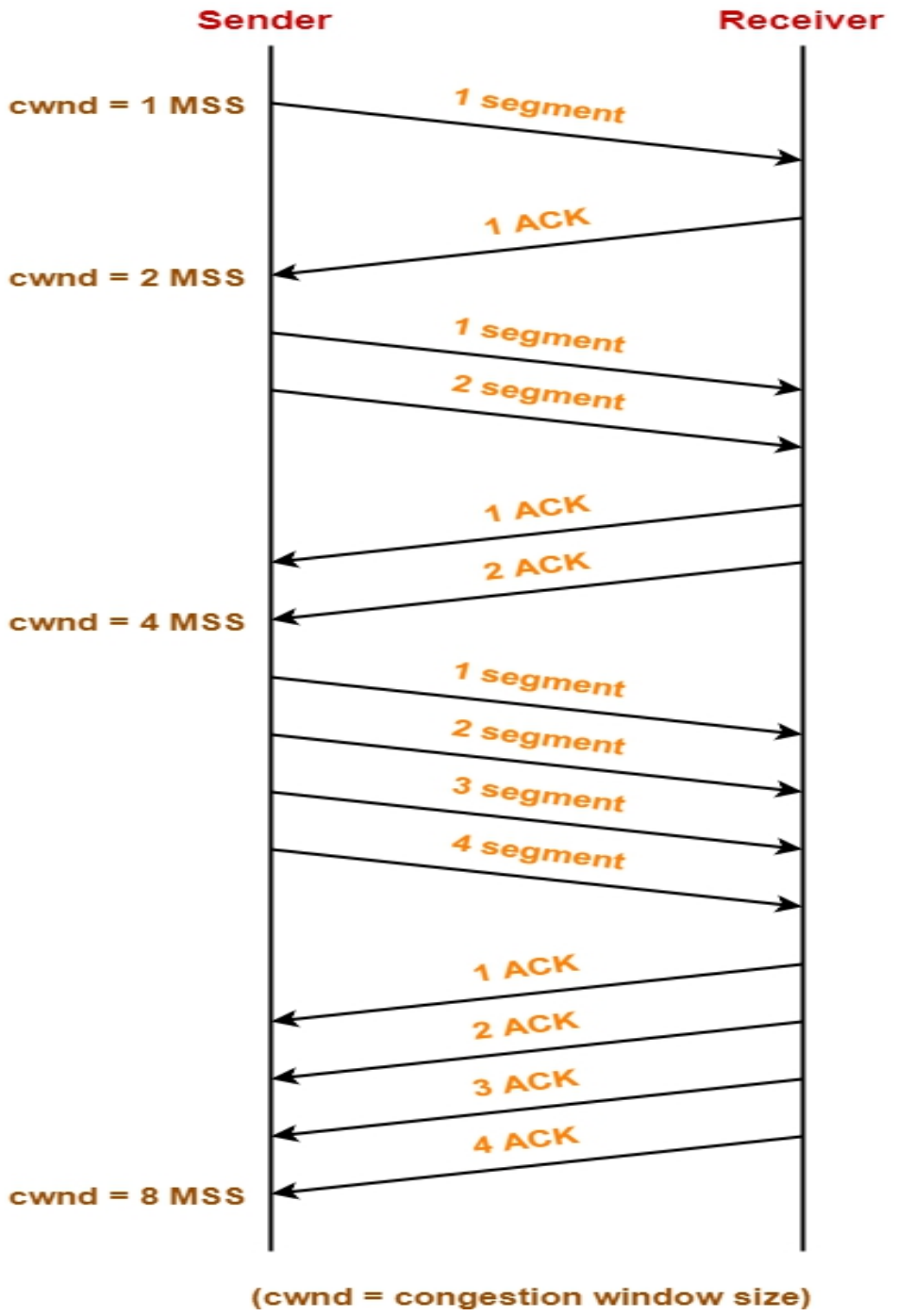
- Initially, sender sets congestion window size = Maximum Segment Size (1 MSS).
- After receiving each acknowledgment, sender increases the congestion window size by 1 MSS.
- In this phase, the size of congestion window increases exponentially.

Congestion window size = Congestion window size + Maximum segment size

After 1 round trip time, congestion window size = $(2)^1$ = 2 MSS
After 2 round trip time, congestion window size = $(2)^2$ = 4 MSS
After 3 round trip time, congestion window size = $(2)^3$ = 8 MSS and so on.

This phase continues til the congestion window size reaches the slow start threshold.

Threshold = (Receiver window size / Maximum Segment Size) / 2



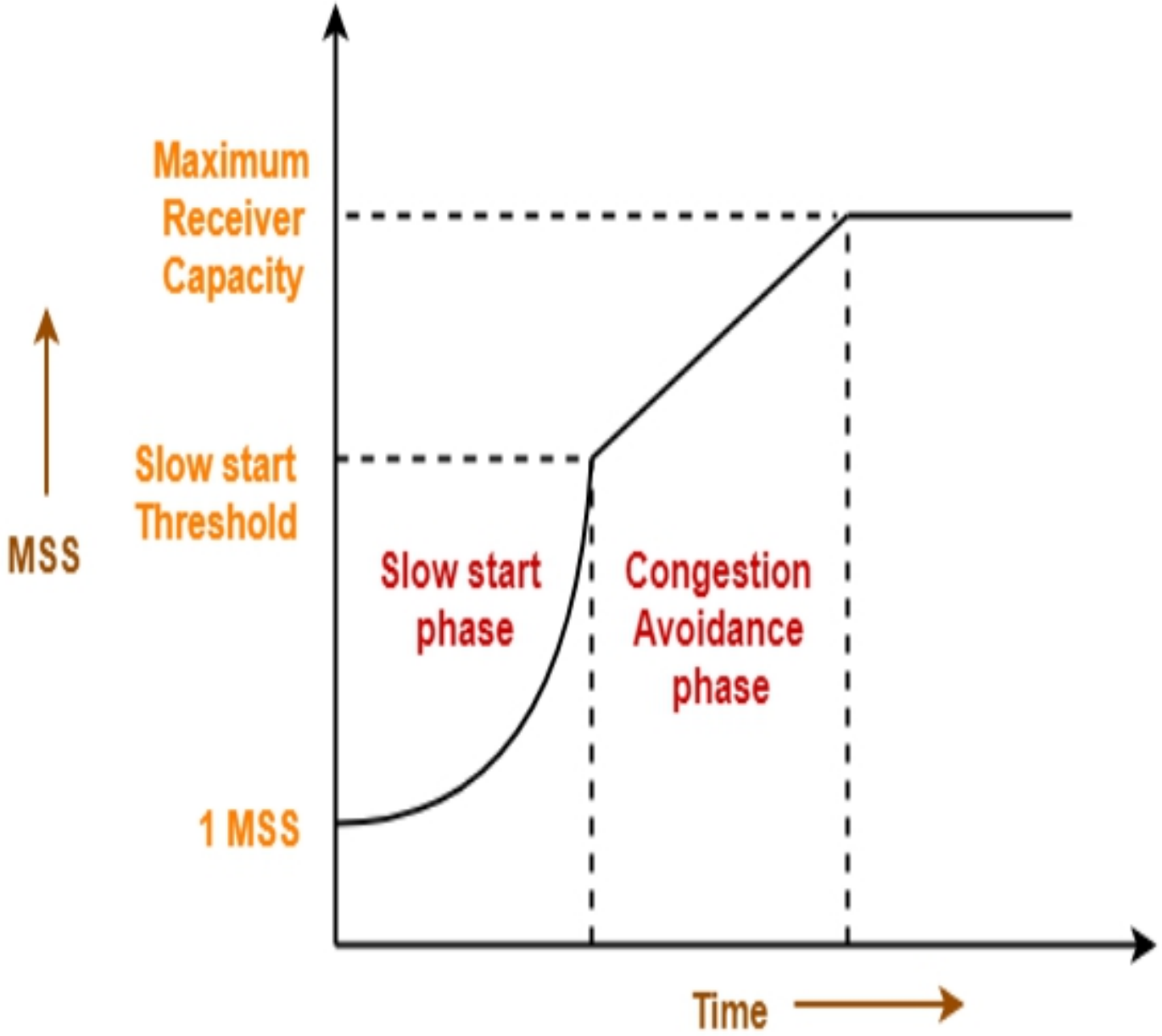
Congestion Avoidance Phase-

After reaching the threshold,

- Sender increases the congestion window size linearly to avoid the congestion.
- On receiving each acknowledgement, sender increments the congestion window size by 1.

Congestion window size = Congestion window size + 1

This phase continues until the congestion window size becomes equal to the receiver window size.



Congestion Detection Phase-

When sender detects the loss of segments, it reacts in different ways depending on how the loss is detected-

Case-01: Detection On Time Out-

- ✓ Time Out Timer expires before receiving the acknowledgement for a segment.
- ✓ This case suggests the stronger possibility of congestion in the network.
- ✓ There are chances that a segment has been dropped in the network.

In this case, sender reacts by-

- ✓ Setting the slow start threshold to half of the current congestion window size.
- ✓ Decreasing the congestion window size to 1 MSS.
- ✓ Resuming the slow start phase.

Case-02: Detection On Receiving 3 Duplicate Acknowledgements-

- ✓ Sender receives 3 duplicate acknowledgements for a segment.
- ✓ This case suggests the weaker possibility of congestion in the network.
- ✓ There are chances that a segment has been dropped but few segments sent later may have reached.

In this case, sender reacts by-

- ✓ Setting the slow start threshold to half of the current congestion window size.
- ✓ Decreasing the congestion window size to slow start threshold.
- ✓ Resuming the congestion avoidance phase.

Q. Consider the effect of using slow start on a line with a 10 msec RTT and no congestion. The receiver window is 24 KB and the maximum segment size is 2 KB. How long does it take before the first full window can be sent?

Given-

Receiver window size = 24 KB

Maximum Segment Size = 2 KB

RTT = 10 msec

Receiver Window Size-

Receiver window size in terms of MSS

= Receiver window size / Size of 1 MSS

= 24 KB / 2 KB = 12 MSS

Slow Start Threshold-

Slow start Threshold

= Receiver window size / 2

= 12 MSS / 2 = 6 MSS

Slow Start Phase-

- Window size at the start of 1st transmission = 1 MSS
- Window size at the start of 2nd transmission = 2 MSS
- Window size at the start of 3rd transmission = 4 MSS
- Window size at the start of 4th transmission = 6 MSS

Since the threshold is reached,
so it marks the end of slow start phase.

Now, congestion avoidance phase begins.

Congestion Avoidance Phase-

- Window size at the start of 5th transmission = 7 MSS
- Window size at the start of 6th transmission = 8 MSS
- Window size at the start of 7th transmission = 9 MSS
- Window size at the start of 8th transmission = 10 MSS
- Window size at the start of 9th transmission = 11 MSS
- Window size at the start of 10th transmission = 12 MSS

From here,

- Window size at the end of 9th transmission
  or at the start of 10th transmission is 12 MSS.
- Thus, 9 RTT's will be taken before the first full window can be sent.

So,

Time taken before the first full window is sent

= 9 RTT's

= 9 x 10 msec = 90 msec

Q. Suppose that the TCP congestion window is set to 18 KB and a time out occurs. How big will the window be if the next four transmission bursts are all successful? Assume that the MSS is 1 KB.

Congestion Window Size-

Congestion window size in terms of MSS

= 18 KB / Size of 1 MSS

= 18 KB / 1 KB = 18 MSS

Reaction Of TCP On Time Out-

TCP reacts by-

- Setting the slow start threshold to half of the current congestion window size.
- Decreasing the congestion window size to 1 MSS.
- Resuming the slow start phase.

So now,

- Slow start threshold = 18 MSS / 2 = 9 MSS
- Congestion window size = 1 MSS

Slow Start Phase-

- Window size at the start of 1^{st} transmission = 1 MSS
- Window size at the start of 2^{nd} transmission = 2 MSS
- Window size at the start of 3^{rd} transmission = 4 MSS
- Window size at the start of 4^{th} transmission = 8 MSS
- Window size at the start of 5^{th} transmission = 9 MSS

Thus. after 4 successful transmissions.
window size will be 9 MSS or 9 KB.

Q. On a TCP connection, current congestion window size is 4 KB. The window advertised by the receiver is 6 KB. The last byte sent by the sender is 10240 and the last byte acknowledged by the receiver is 8192.

Find the current window size at the sender

Find the amount of free space in the sender window

Part-01:

Sender window size

= min (Congestion window size, Receiver window size)

= min(4KB , 6KB)

= 4 KB

= 4096 B

Part-02:

Given-

- Last byte acknowledged by the receiver = 8192
- Last byte sent by the sender = 10240

From here,

- It means bytes from 8193 to 10240 are still present in the sender's window.
- These bytes are waiting for their acknowledgement.
- Total bytes present in sender's window = 10240 – 8193 + 1 = 2048 bytes.

From here,

Amount of free space in sender's window currently

= 4096 bytes – 2048 bytes

END