Transport Layer

Prof.Mohammed Juned

- **5.1 The Transport Service: Transport service primitives, Berkeley Sockets, Connection management (Handshake), UDP, TCP, TCP state transition, TCP timers**

  5.2 TCP Flow control (sliding Window), TCP Congestion Control: Slow Start

- TCP Primitives : A service is specified by a set of primitives. A primitive means operation. To access the service a user process can access these primitives. These primitives are different for connection-oriented service and connectionless service.
- There are five types of service primitives:

1. **LISTEN :** When a server is ready to accept an incoming connection it executes the LISTEN primitive. It blocks waiting for an incoming connection.
2. **CONNECT** : It connects the server by establishing a connection. Response is awaited.
3. **RECIEVE**: Then the RECIEVE call blocks the server.
4. **SEND** : Then the client executes SEND primitive to transmit its request followed by the execution of RECIEVE to get the reply. Send the message.
5. **DISCONNECT** : This primitive is used for terminating the connection. After this primitive one can’t send any message.
6. When the client sends DISCONNECT packet then the server also sends the DISCONNECT packet to acknowledge the client. When the server package is received by client then the process is terminated.

- **Connection Oriented Service Primitives**
- The types of primitives for Connection Oriented Service :

CONNECT	This primitive makes a connection
DATA, DATA-ACKNOWLEDGE, EXPEDITED-DATA	Data and information is sent using thus primitive
RESET	Primitive for reseting the connection

- **Connectionless Oriented Service Primitives**
- There are 2 types of primitives for Connectionless Oriented Service:

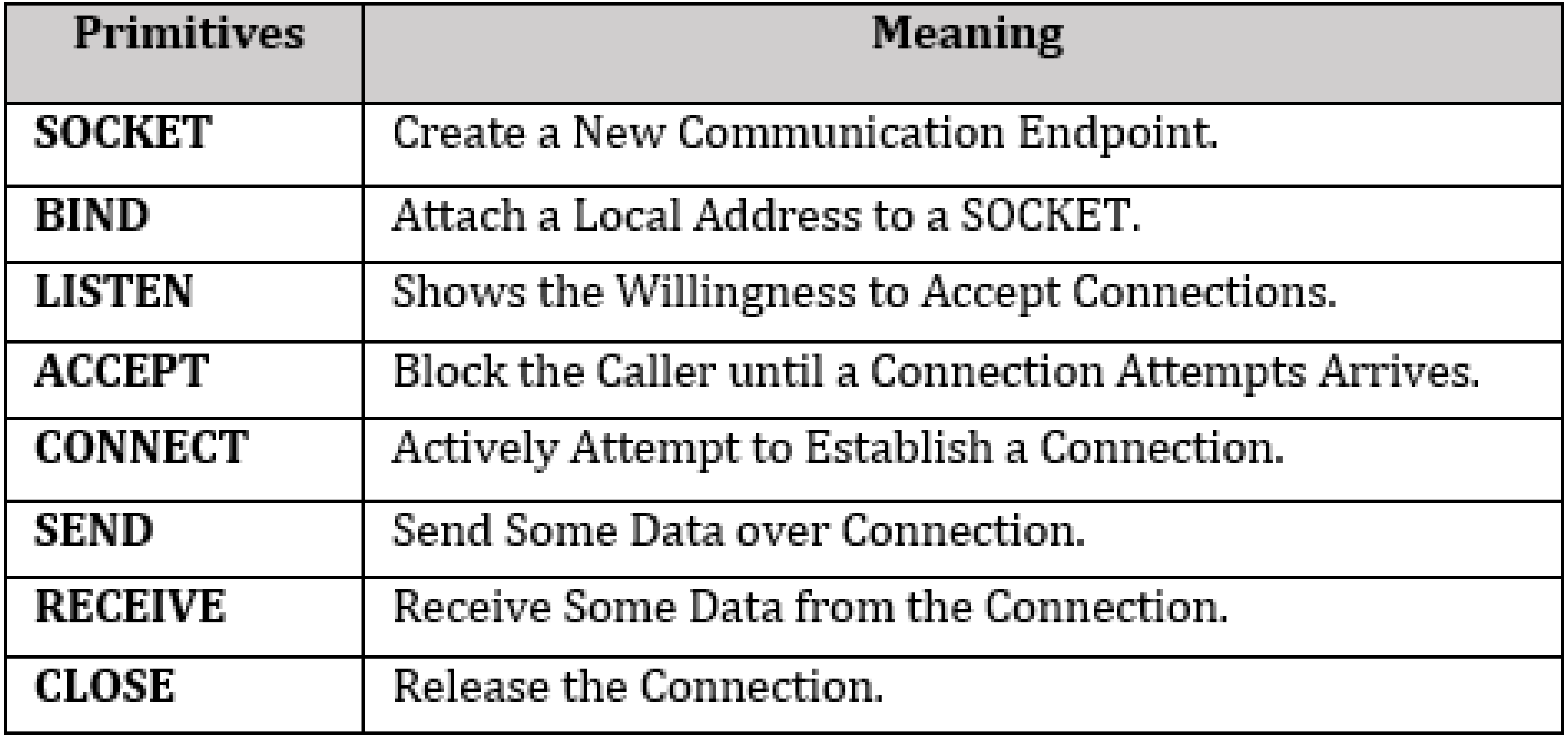
UNIDATA	This primitive sends a packet of data
FACILITY, REPORT	Primitive for enquiring about the performance of the network, like delivery statistics.

- **Socket:**

1. Sockets are a service provided by transport layer.
2. A socket is one endpoint of a two-way communication link between two programs running on the network.

- **Berkeley Socket:**

1. Berkeley sockets is an application programming interface (API) for Internet sockets.
2. It is used for inter-process communication (IPC).
3. It is commonly implemented as a library of linkable modules.



Primitives	Meaning
SOCKET	Create a New Communication Endpoint.
BIND	Attach a Local Address to a SOCKET.
LISTEN	Shows the Willingness to Accept Connections.
ACCEPT	Block the Caller until a Connection Attempts Arrives.
CONNECT	Actively Attempt to Establish a Connection.
SEND	Send Some Data over Connection.
RECEIVE	Receive Some Data from the Connection.
CLOSE	Release the Connection.

- **Socket Programming:**
- **I) Server side:**
- Server start-up executes SOCKET, BIND & LISTEN primitives.
- LISTEN primitive allocate queue for multiple simultaneous clients.
- Then it use ACCEPT to suspend server until request.
- When client request arrives: ACCEPT returns.
- Start new socket (thread or process) with same properties as original, this handles the request, server goes on waiting on original socket.
- If new request arrives while spawning thread for this one, it is queued.
- If queue full it is refused.

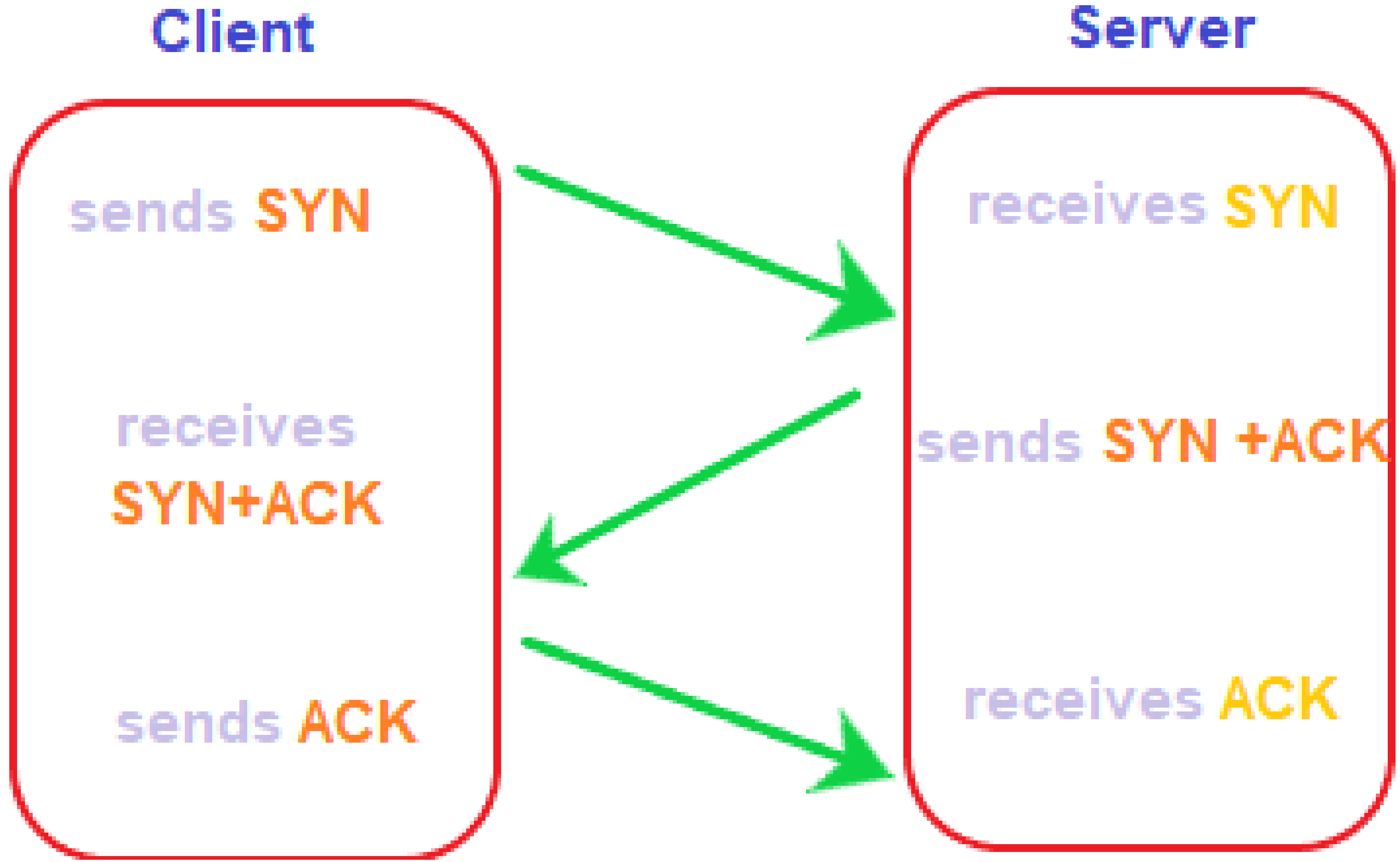
- **I) Client side:**
- It uses SOCKET primitives to create.
- Then use CONNECT to initiate connection process.
- When this returns the socket is open.
- Both sides can now SEND, RECEIVE.
- Connection not released until both sides do CLOSE.
- Typically, client does it, server acknowledges.

• TCP Connection Management :

- Handshake refers to the process to establish connection between the client and server. Handshake is simply defined as the process to establish a communication link.

- To transmit a packet, TCP needs a three-way handshake before it starts sending data.
- The reliable communication in TCP is termed as **PAR** (Positive Acknowledgement Re-transmission).

- When a sender sends the data to the receiver, it requires a positive acknowledgement from the receiver confirming the arrival of data. If the acknowledgement has not reached the sender, it needs to resend that data. The positive acknowledgement from the receiver establishes a successful connection.



Client

Server

sends SYN

receives SYN

sends SYN +ACK

receives SYN+ACK

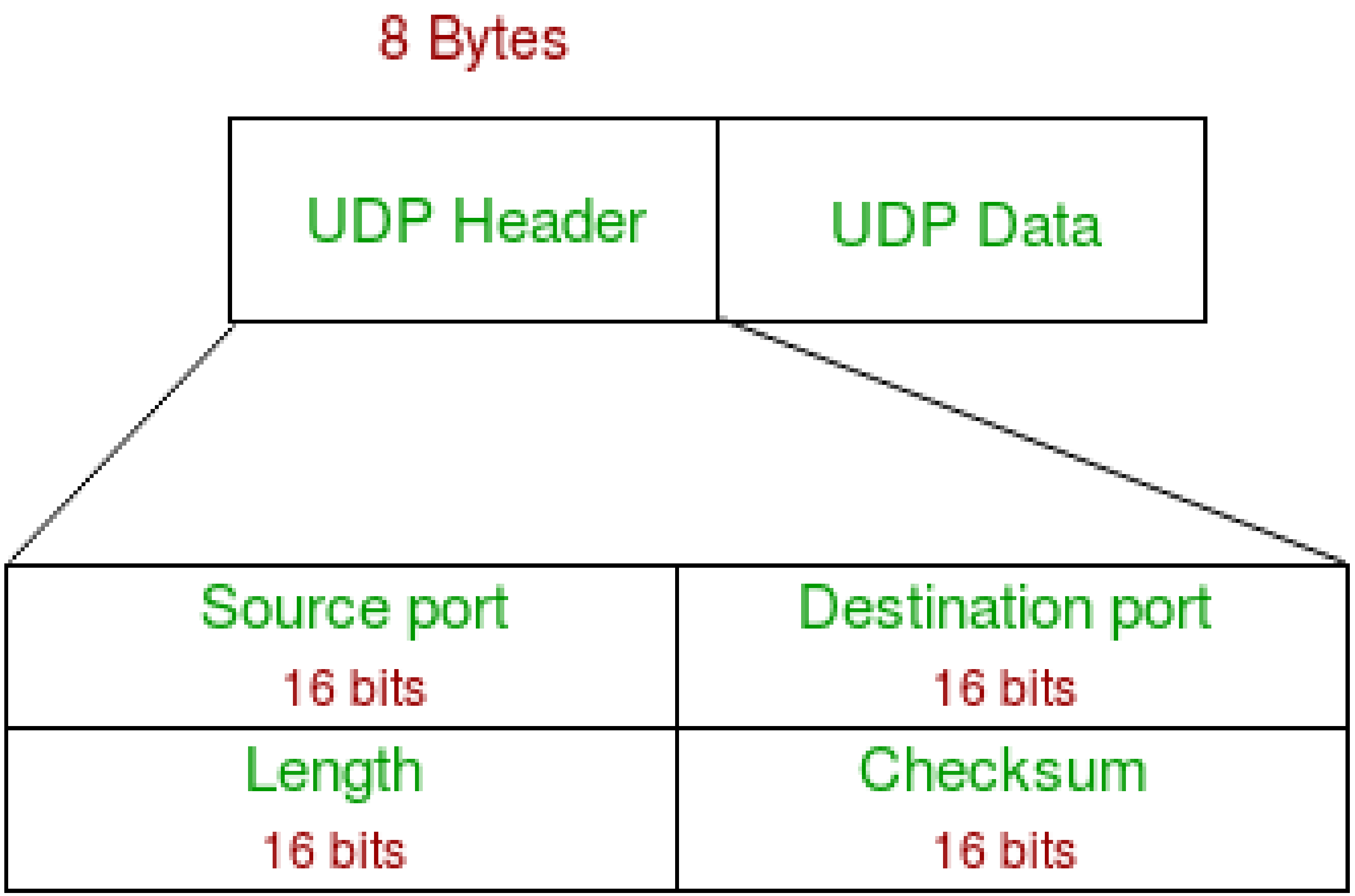
sends ACK

receives ACK

- **Step 1: SYN**
- SYN is a segment sent by the client to the server.
- It acts as a **connection request** between the client and server.
- It informs the server that the client wants to establish a connection. Synchronizing sequence numbers also helps synchronize sequence numbers sent between any two devices, where the same SYN segment asks for the sequence number with the connection request.
- **Step 2: SYN-ACK**
- It is an SYN-ACK segment or an SYN + ACK segment sent by the server.
- The ACK segment informs the client that the server has received the connection request and it is ready to build the connection. The SYN segment informs the sequence number with which the server is ready to start with the segments.

- **Step 3: ACK**
- ACK (Acknowledgment) is the last step before establishing a successful TCP connection between the client and server.
- The ACK segment is sent by the client as the response of the received ACK and SN from the server. It results in the establishment of a reliable data connection.
- After these three steps, the client and server are ready for the data communication process.
- TCP connection and termination are full-duplex, which means that the data can travel in both the directions simultaneously.

- **UDP :**
- **User Datagram Protocol (UDP)** is a Transport Layer protocol. UDP is a part of the Internet Protocol suite, referred to as UDP/IP suite.
- Unlike TCP, it is an **unreliable and connectionless protocol.** So, there is no need to establish a connection prior to data transfer.
- For **real-time 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 saves bandwidth.
  User Datagram Protocol (UDP) is more efficient in terms of both latency and bandwidth.
- UDP header is an **8-bytes** fixed and simple header, while for TCP it may vary from 20 bytes to 60 bytes.
- The **first 8 Bytes** contains all necessary header information and the remaining part consist of data. UDP port number fields are each 16 bits long, therefore the range for port numbers is defined from 0 to 65535; port number 0 is reserved. Port numbers help to distinguish different user requests or processes.



8 Bytes

UDP Header

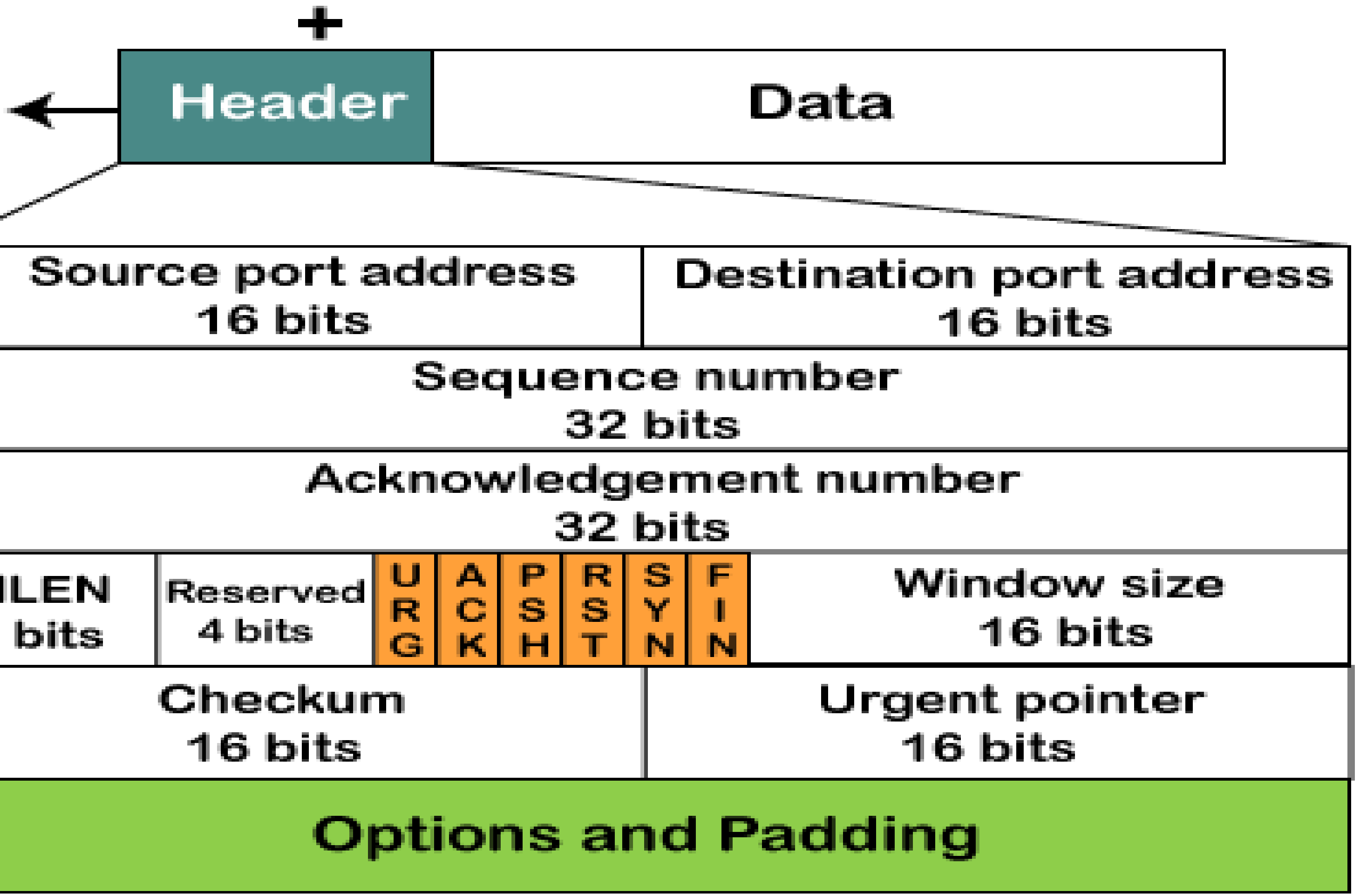
UDP Data

Source port 16 bits	Destination port 16 bits
Length 16 bits	Checksum 16 bits

1. **Source Port:** Source Port is a 2 Byte long field used to identify the port number of the source.
2. **Destination Port:** It is a 2 Byte long field, used to identify the port of the destined packet.
3. **Length:** Length is the length of UDP including the header and the data. It is a 16-bits field.
4. **Checksum:** Checksum is 2 Bytes long field. It is the 16-bit one’s complement of the one’s complement sum of the UDP header.
5. **Notes** – Unlike TCP, the Checksum calculation is not mandatory in UDP. No Error control or flow control is provided by UDP. Hence UDP depends on IP and ICMP for error reporting.

- **TCP**
- TCP stands for **Transmission Control Protocol**. It is a transport layer protocol that facilitates the transmission of packets from source to destination.
- It is a connection-oriented protocol that means it establishes the connection prior to the communication that occurs between the computing devices in a network.
- This protocol is used with an IP protocol, so together, they are referred to as a TCP/IP.
- The main functionality of the TCP is to take the data from the application layer. Then it divides the data into a several packets, provides numbering to these packets, and finally transmits these packets to the destination.
- The TCP, on the other side, will reassemble the packets and transmits them to the application layer.

TCP header format



+

Header

Data

Source port address
16 bits

Destination port address
16 bits

Sequence number
32 bits

Acknowledgement number
32 bits

HLEN
4 bits

Reserved
4 bits

URG

ACK

PSH

RST

SYN

FIN

Window size
16 bits

Checkum
16 bits

Urgent pointer
16 bits

Options and Padding

- **Source port:** It defines the port of the application, which is sending the data. So, this field contains the source port address, which is 16 bits.
- **Destination port:** It defines the port of the application on the receiving side. So, this field contains the destination port address, which is 16 bits.
- **Sequence number:** This field contains the sequence number of data bytes in a particular session.
- **Acknowledgment number:** When the ACK flag is set, then this contains the next sequence number of the data byte and works as an acknowledgment for the previous data received. For example, if the receiver receives the segment number 'x', then it responds 'x+1' as an acknowledgment number.

- **HLEN:** It specifies the length of the header indicated by the 4-byte words in the header. The size of the header lies between 20 and 60 bytes.
- Therefore, the value of this field would lie between 5 and 15.
- **Reserved:** It is a 4-bit field reserved for future use, and by default, all are set to zero.
- **Flags**

There are six control bits or flags:

  - **URG:** It represents an urgent pointer. If it is set, then the data is processed urgently.
  - **ACK:** If the ACK is set to 0, then it means that the data packet does not contain an acknowledgment.
  - **PSH:** If this field is set, then it requests the receiving device to push the data to the receiving application without buffering it.
  - **RST:** If it is set, then it requests to restart a connection.
  - **SYN:** It is used to establish a connection between the hosts.
  - **FIN:** It is used to release a connection, and no further data exchange will happen.

- **Window size :**
- It is a 16-bit field. It contains the size of data that the receiver can accept.
- This field is used for the flow control between the sender and receiver and determines the amount of buffer allocated by the receiver for a segment. The value of this field is determined by the receiver.
- **Checksum** : It is a 16-bit field. This field is optional in UDP, but in the case of TCP/IP, this field is mandatory.
- **Urgent pointer :** It is a pointer that points to the urgent data byte if the URG flag is set to 1. It defines a value that will be added to the sequence number to get the sequence number of the last urgent byte.
- **Options :** It provides additional options. The optional field is represented in 32-bits. If this field contains the data less than 32-bit, then padding is required to obtain the remaining bits.

TCP Timers :

- TCP uses several timers to ensure that excessive delays are not encountered during communications. Several of these timers are elegant, handling problems that are not immediately obvious at first analysis.
- Each of the timers used by TCP is examined in the following sections, which reveal its role in ensuring data is properly sent from one connection to another.
- **TCP implementation uses four timers –**

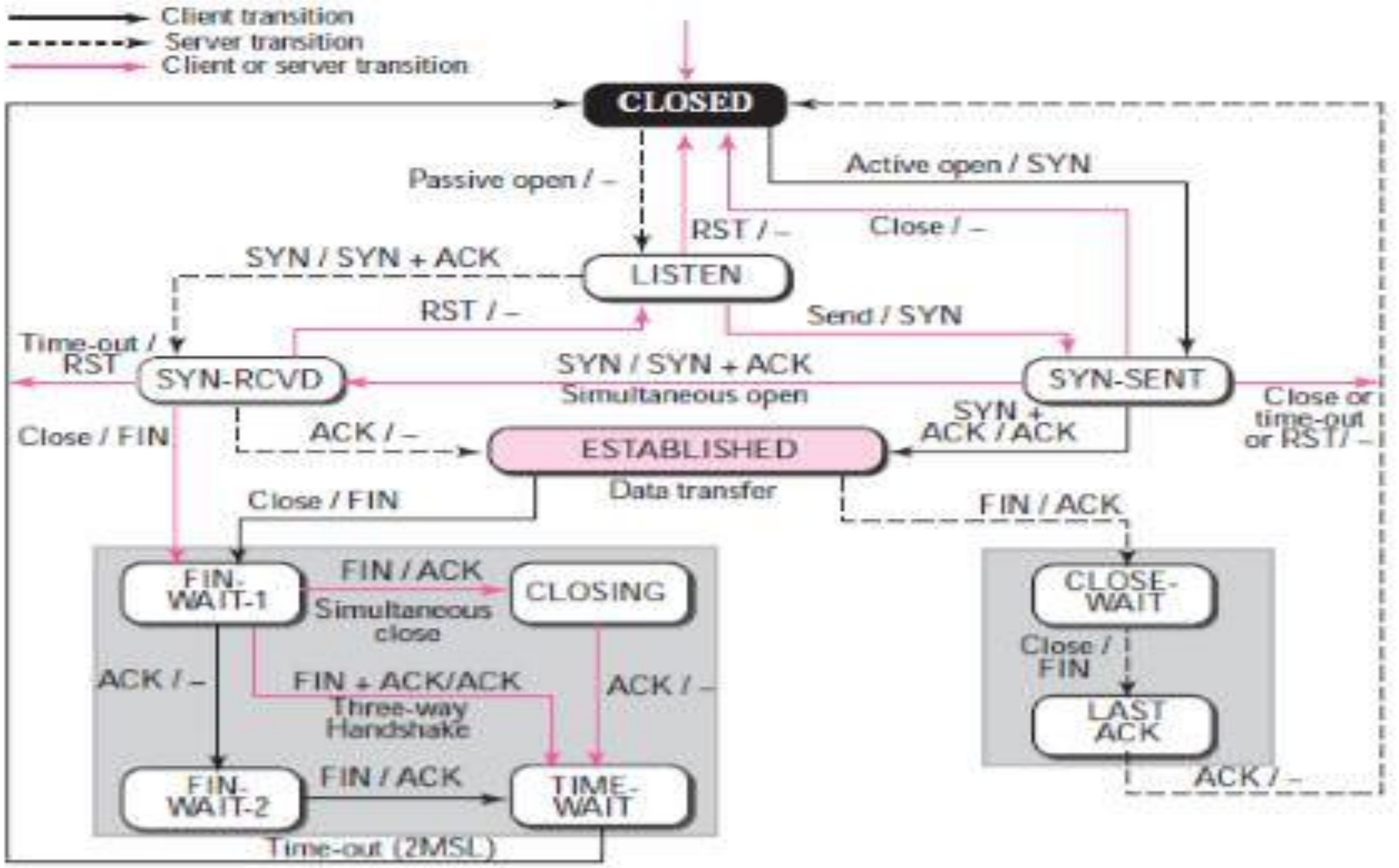
- TCP uses different types of timer to control and management various tasks:
- **Keep-alive timer:**
- This timer is used to check the integrity and validity of a connection.
- When keep-alive time expires, the host sends a probe to check if the connection still exists.
- **Retransmission timer:**
- This timer maintains stateful session of data sent.
- If the acknowledgement of sent data does not receive within the Retransmission time, the data segment is sent again.

Persist timer:

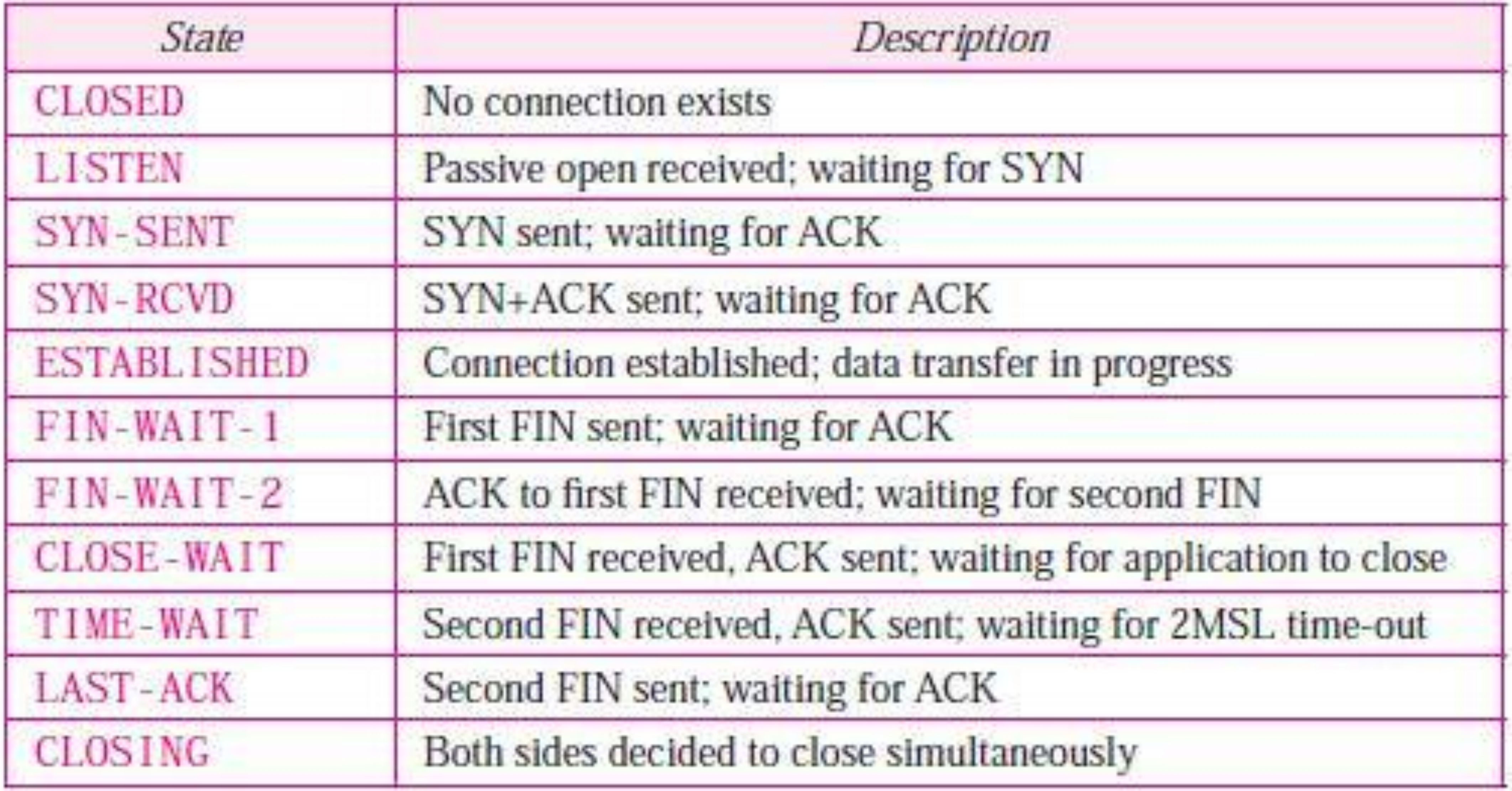
- TCP session can be paused by either host by sending Window Size 0.
- To resume the session a host needs to send Window Size with some larger value.
- If this segment never reaches the other end, both ends may wait for each other for infinite time.
- When the Persist timer expires, the host re-sends its window size to let the other end know.
- Persist Timer helps avoid deadlocks in communication.

Timed-Wait:

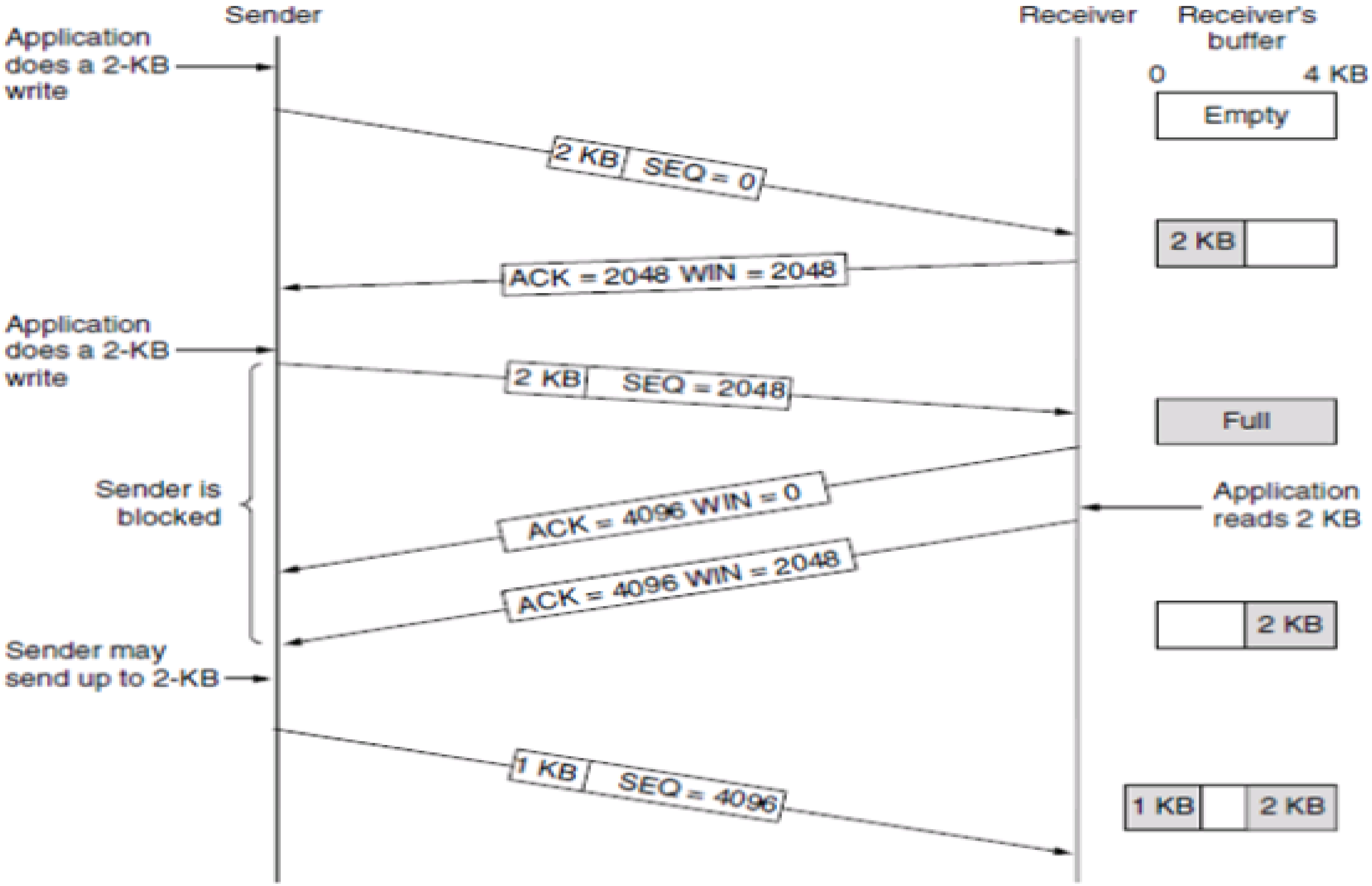
- After releasing a connection, either of the hosts waits for a Timed-Wait time to terminate the connection completely.
- This is in order to make sure that the other end has received the acknowledgement of its connection termination request.
- Timed-out can be a maximum of 240 seconds (4 minutes).


Client transition
Server transition
Client or server transition
CLOSED
Passive open / –
RST / –
Close / –
Active open / SYN
SYN / SYN + ACK
LISTEN
RST / –
Send / SYN
Time-out / RST
SYN-RCVD
SYN / SYN + ACK
Simultaneous open
SYN-SENT
Close or time-out or RST/ –
Close / FIN
ACK / –
ESTABLISHED
SYN + ACK / ACK
Data transfer
Close / FIN
FIN / ACK
FIN-WAIT-1
FIN / ACK
Simultaneous close
CLOSING
CLOSE-WAIT
ACK / –
FIN + ACK/ACK
Three-way Handshake
ACK / –
Close / FIN
LAST ACK
FIN-WAIT-2
FIN / ACK
TIME-WAIT
ACK / –
Time-out (2MSL)

- The figure shows the two FSMs used by the TCP client and server combined in one diagram. The ovals represent the states. The transition from one state to another is shown using directed lines.
- The dotted black lines in the figure represent the transition that a server normally goes through; the solid black lines show the transitions that a client normally goes through.



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



Window management in TCP

- Window management in TCP decouples the issues of acknowledgement of the correct receipt of segments and receiver buffer allocation.
- For example, suppose the receiver has a 4096-byte buffer, as shown in Fig. below.
- If the sender transmits a 2048-byte segment that is correctly received, the receiver will acknowledge the segment.
- However, since it now has only 2048 bytes of buffer space (until the application removes some data from the buffer), it will advertise a window of 2048 starting at the next byte expected

- Now the sender transmits another 2048 bytes, which are acknowledged, but the advertised window is of size 0.
- The sender must stop until the application process on the receiving host has removed some data from the buffer, at which time TCP can advertise a larger window and more data can be sent.
- When the window is 0, the sender may not normally send segments, with two exceptions.
- First, urgent data may be sent, for example, to allow the user to kill the process running on the remote machine. Second, the sender may send a 1-byte segment to force the receiver to re-announce the next byte expected and the window size. This packet is called a window probe.

- The TCP standard explicitly provides this option to prevent deadlock if a window update ever gets lost.
- Senders are not required to transmit data as soon as they come in from the application.
- Neither are receivers required to send acknowledgements as soon as possible. For example, in Fig. above, when the first 2 KB of data came in, TCP, knowing that it had a 4-KB window, would have been completely correct in just buffering the data until another 2 KB came in, to be able to transmit a segment with a 4-KB payload. This freedom can be used to improve performance.
- Consider a connection to a remote terminal, for example using SSH or telnet, that reacts on every keystroke. In the worst case, whenever a character arrives at the sending TCP entity, TCP creates a 21-byte TCP segment, which it gives to IP to send as a 41-byte IP datagram.

- At the receiving side, TCP immediately sends a 40-byte acknowledgement (20 bytes of TCP header and 20 bytes of IP header).
- Later, when the remote terminal has read the byte, TCP sends a window update, moving the window 1 byte to the right. This packet is also 40 bytes.
- Finally, when the remote terminal has processed the character, it echoes the character for local display using a 41-byte packet.
- In all, 162 bytes of bandwidth are used, and four segments are sent for each character typed. When bandwidth is scarce, this method of doing business is not desirable.

- 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.
- **Congestion Control-**
- 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-

1. Receiver window size
2. Congestion window size

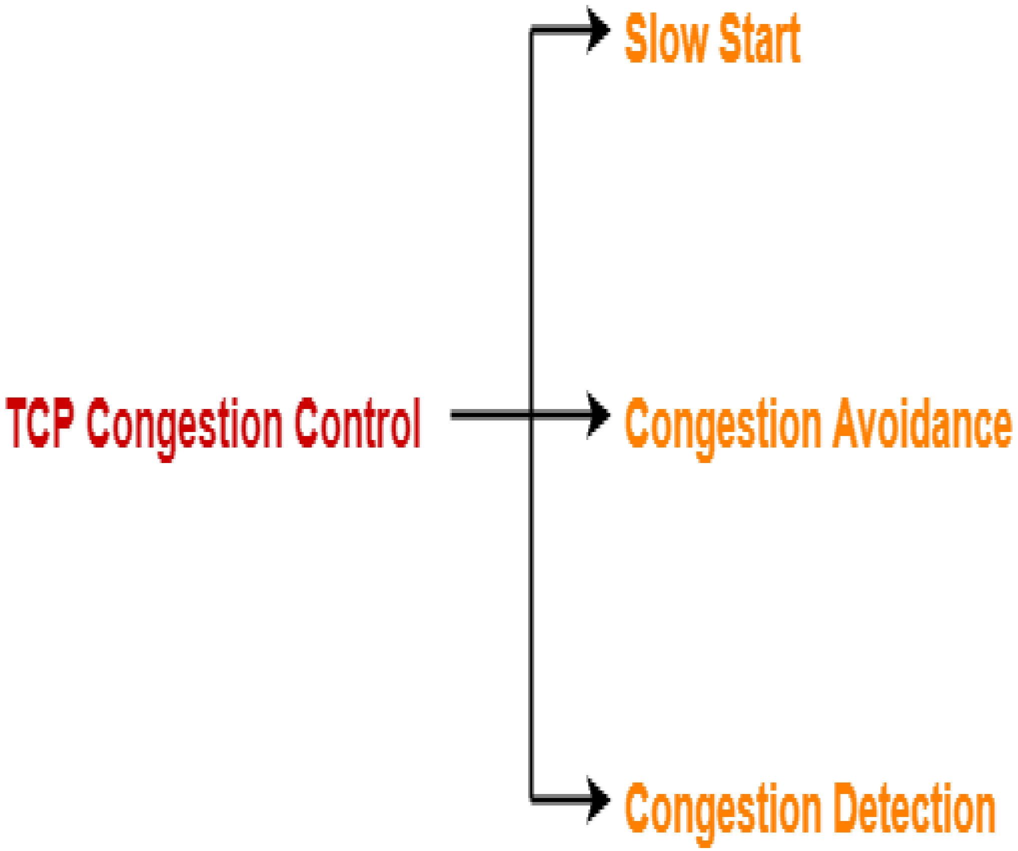
- **<u>1. Receiver Window Size-</u>**
  - Receiver window size is an advertisement of-
  - “How much data (in bytes) the receiver can receive without acknowledgement?”

- 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**.

- **2. Congestion Window-**

- 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.

- Sender window size = Minimum (Receiver window size, Congestion window size)
- **TCP Congestion Policy-**
- TCP’s general policy for handling congestion consists of following three phases-



TCP Congestion Control

Slow Start

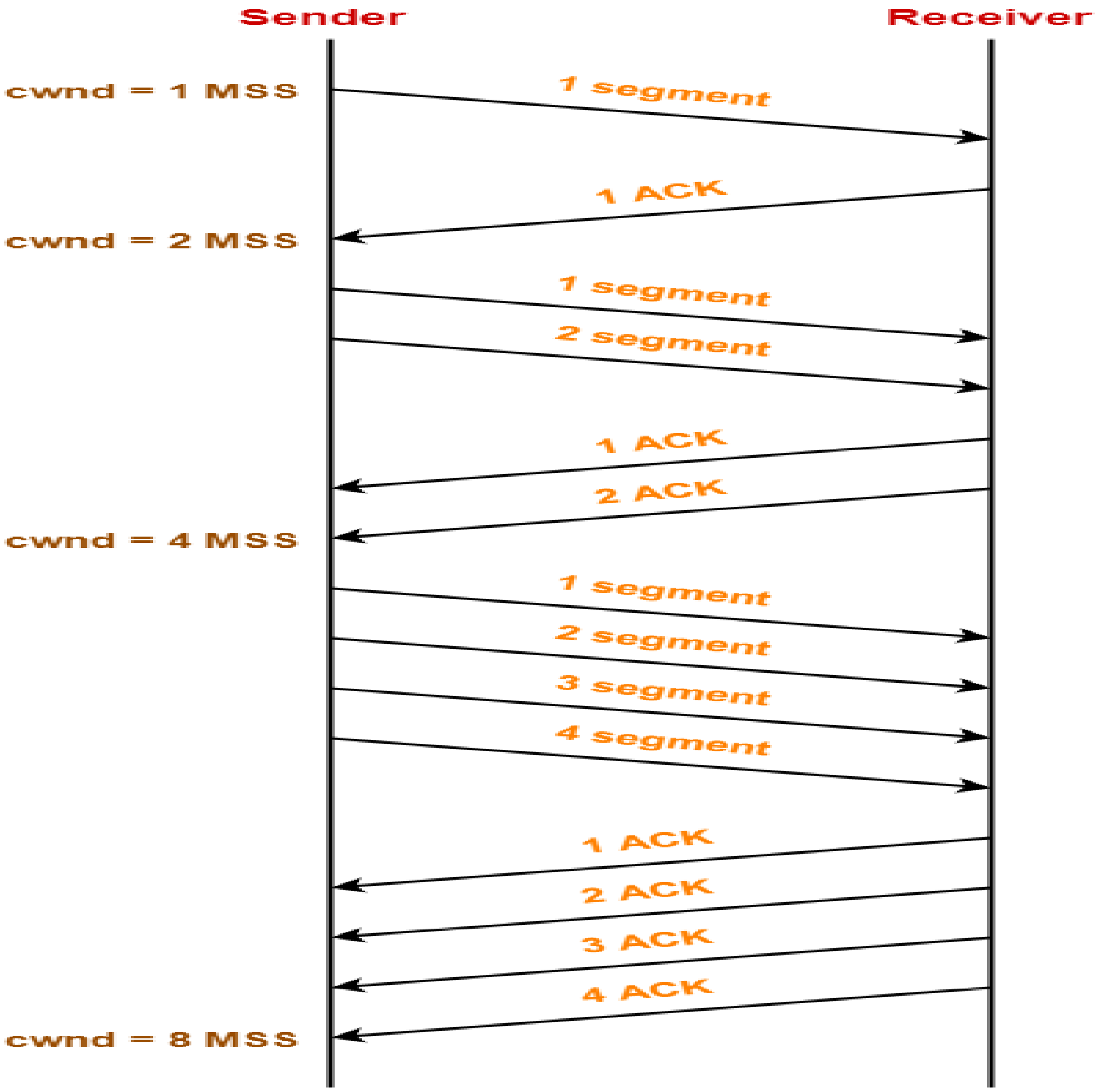
Congestion Avoidance

Congestion Detection

- **1. 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



(cwnd = congestion window size)

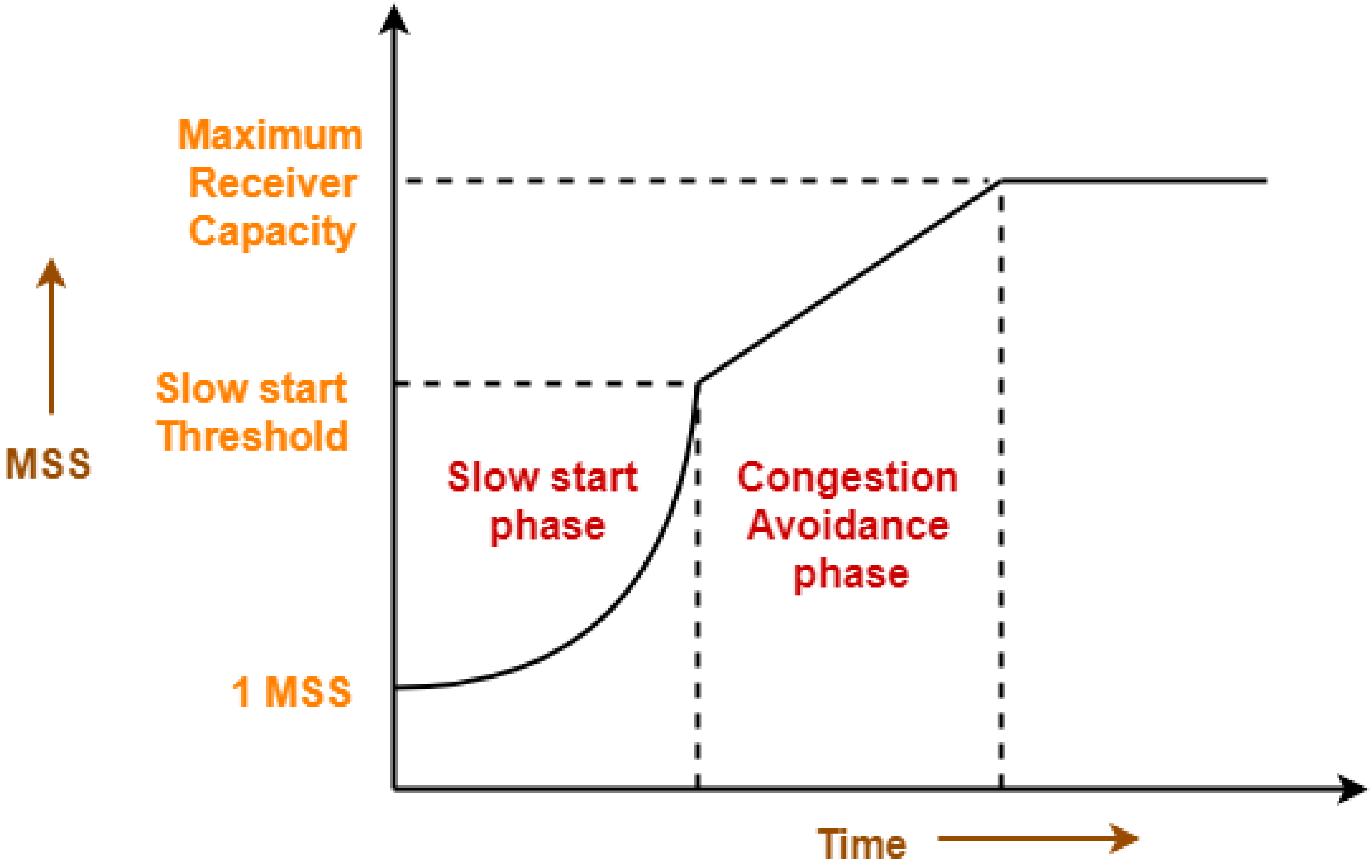
- After 1 round trip time, congestion window size = $(2)^1$ = 2 MSS
- After 2 round trip time, congestion window size = $(2)^2$ = 4 MSS
- After 3 round trip time, congestion window size = $(2)^3$ = 8 MSS and so on.

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

- Threshold
- = Maximum number of TCP segments that receiver window can accommodate / 2
- = (Receiver window size / Maximum Segment Size) / 2

- **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


Maximum
Receiver
Capacity
Slow start
Threshold
MSS
1 MSS
Slow start
phase
Congestion
Avoidance
phase
Time

- End Of Module 5