Computer Network(CSC 503)

Shilpa Ingoley

Lecture 33

TCP Connection Management

- TCP is **connection-oriented**.
- A connection-oriented transport protocol establishes a **logical path** between the source and destination.
- **All of the segments** belonging to a message are then sent over this **logical path**.
- Using a **single logical pathway** for the entire message facilitates the acknowledgment process as well as retransmission of damaged or lost frames.
- In TCP, connection-oriented transmission requires three phases:
  - Connection establishment
  - Data transfer
  - Connection termination

Connection Establishment

- TCP transmits data in **full-duplex** mode.
- When two TCPs in two machines are connected, they are able to send segments to each other **simultaneously**.
- This implies that each party must **initialize communication** and get **approval** from the other party before any data are transferred
- The connection establishment in TCP is called **three-way handshaking**

Three-Way Handshaking

- 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**.
- The server TCP is **ready** to accept a connection from **any machine in the world** but 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.

Steps of Three way Handshaking

Step-1:

- 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**.
- The client in chooses a **random number** as the first sequence number and sends this number to the server.
- This sequence number is called the **initial sequence number (ISN).**
- A SYN segment **cannot carry data**, but it **consumes one sequence number**

Contd...

Step-2:

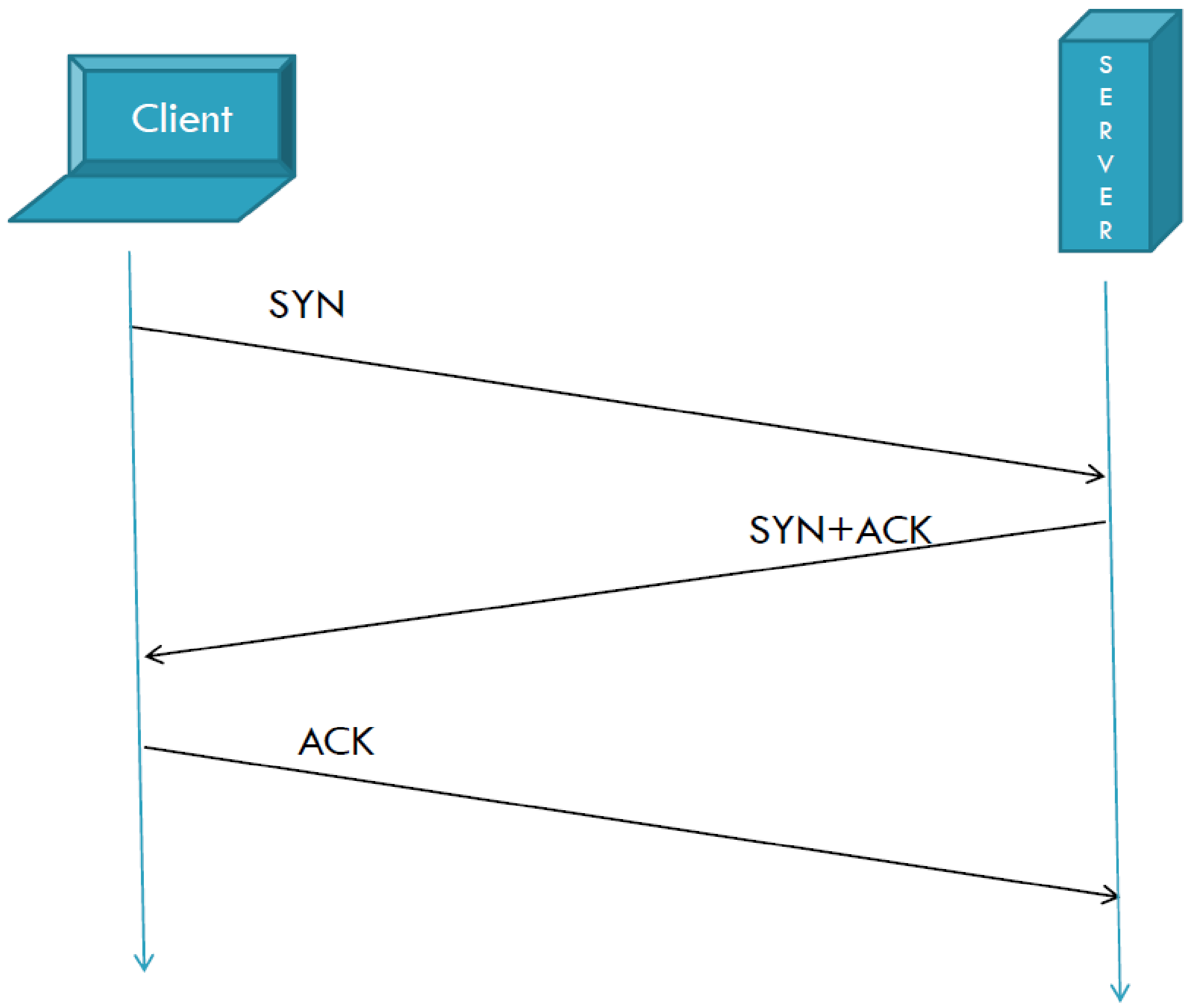
- The server sends the **second** segment, a **SYN + ACK segment** with two flag bits set as: SYN and ACK.
- This segment has a **dual** purpose.
- First, it is a SYN segment for communication in the other direction. The server uses this segment to initialize a **sequence number** for numbering the bytes sent from the server to the client.
- Secondly, 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.
- Because the segment contains an acknowledgment, it also needs to define the receive **window size, rwnd** (to be used by the client),
- Since this segment is playing the role of a SYN segment, it needs to be acknowledged. It, therefore, consumes one sequence number.

Contd…

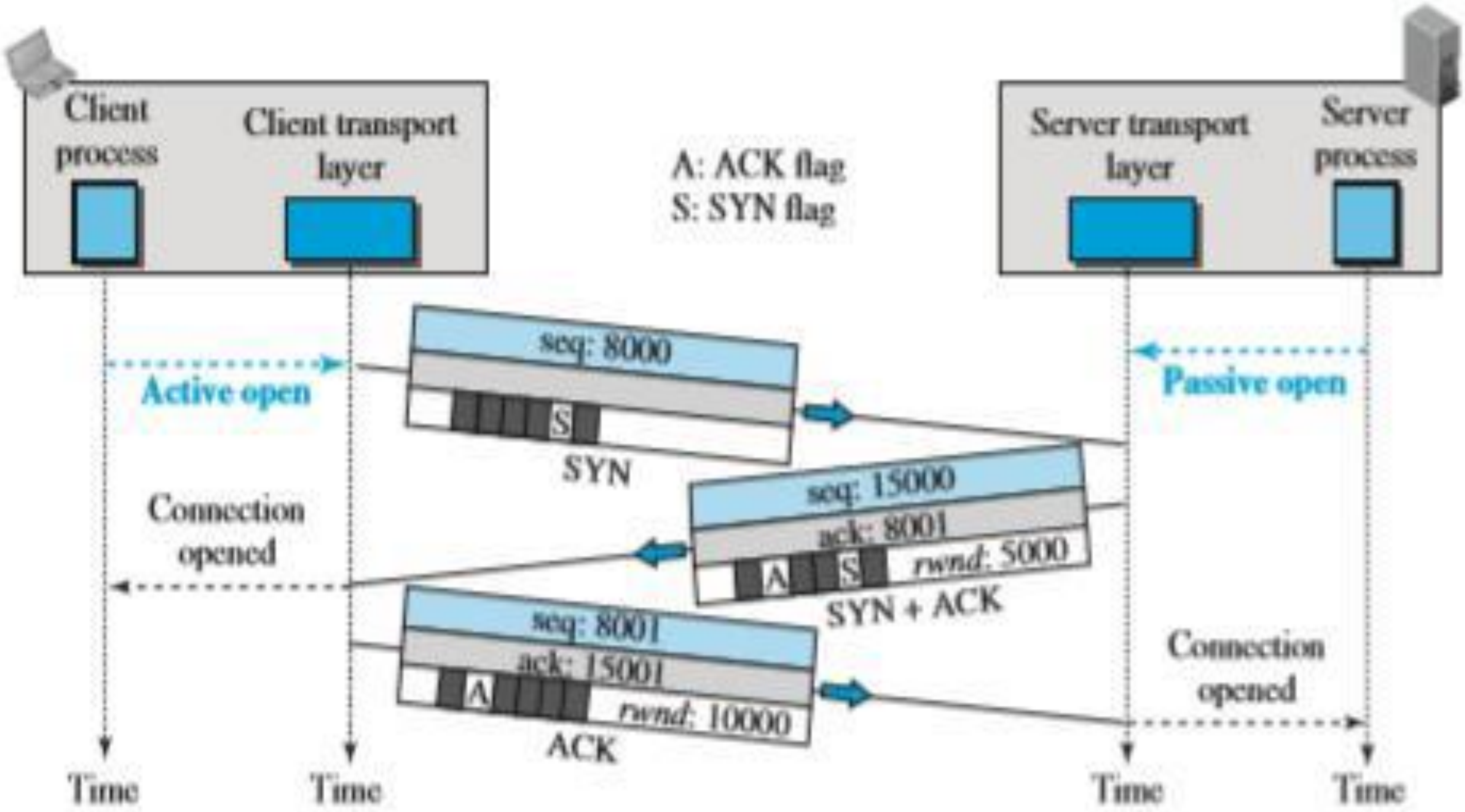
Step-3:

- The client sends the **third** segment. This is an ACK segment.
- It acknowledges the receipt of the **second segment** with the ACK flag and acknowledgment number field.
- The ACK segment **does not consume any sequence numbers** if it does not carry data
- But some implementations allow this **third** segment in the connection phase to carry the first chunk of data from the client.
- In this case, the segment consumes as many sequence numbers as the number of data bytes.

Contd…



Contd…



Client process
Client transport layer
A: ACK flag
S: SYN flag
Server transport layer
Server process
seq: 8000
Active open
S
SYN
Passive open
seq: 15000
ack: 8001
A
S
rwnd: 5000
SYN + ACK
Connection opened
seq: 8001
ack: 15001
A
rwnd: 10000
ACK
Connection opened
Time
Time
Time
Time

Data Transfer

- After connection is established, **bidirectional data transfer** can take place.
- The client and server can send data and acknowledgments in both directions.

Pushing Data (PSH flag):

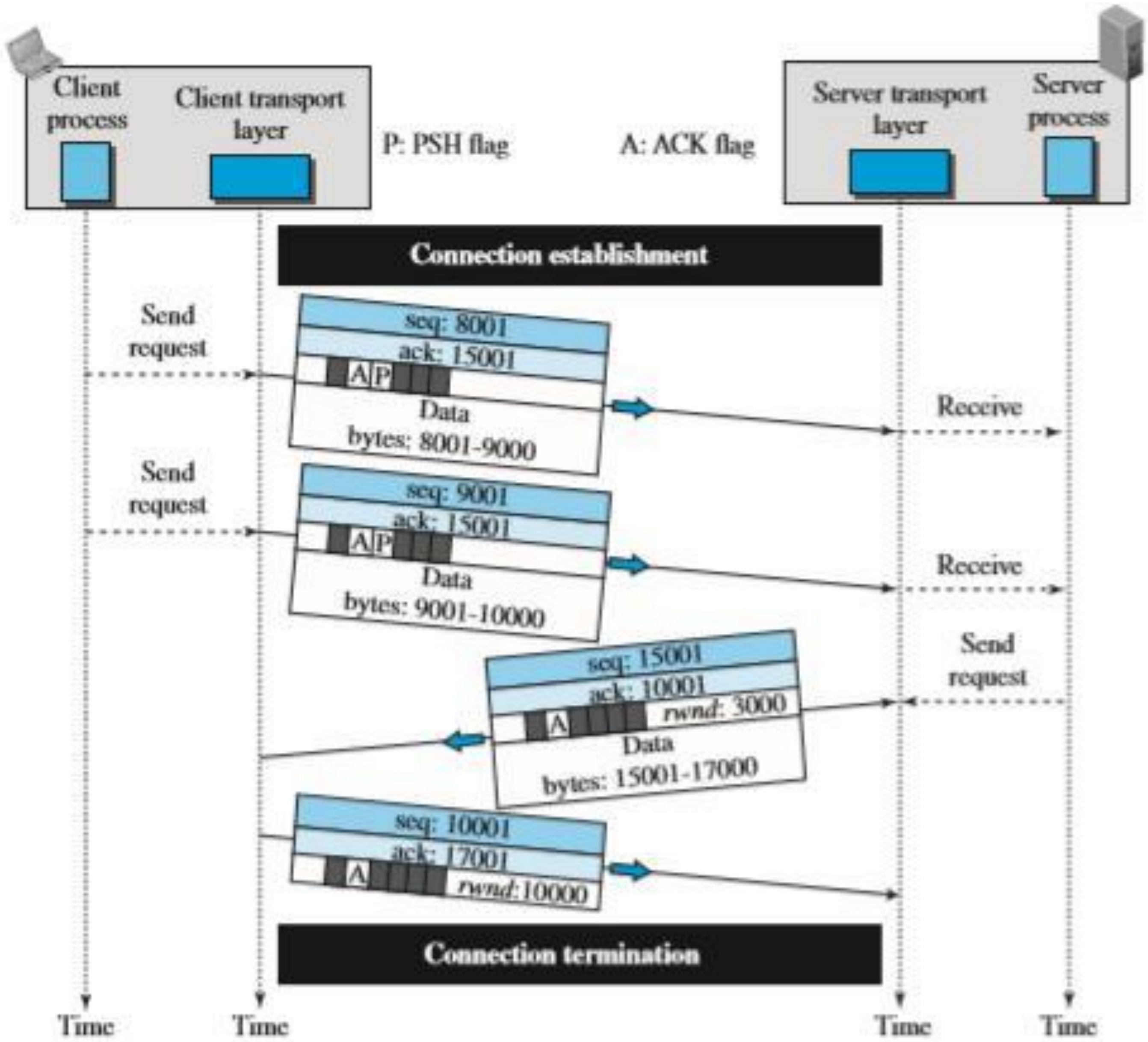
- The data segments sent by the client have the **PSH (push)** flag set so that the server TCP knows to deliver data to the server process as soon as they are received
- This means that the sending **TCP must not wait for the window to be filled**. It must create a segment and **send it immediately**

Contd…

Urgent Data (URG flag)

- There are occasions in which an application program needs to send **urgent** bytes.
- The solution is to send a segment with the **URG** bit set.
- The sending application program tells the sending TCP that the piece of data is urgent
- The sending TCP creates a segment and inserts the urgent data at the beginning of the segment.
- The rest of the segment can contain **normal data** from the **buffer**.
- The urgent pointer field in the header defines the end of the urgent data (the last byte of urgent data)
- Example: If the segment sequence no. is **15000** and the value of **urgent pointer** is **200**, the **first** byte of urgent data is the byte **15000** and the **last** byte is the byte **15200**. The rest of the bytes in the segment are non urgent.

Contd…



Client process
Client transport layer
P: PSH flag
A: ACK flag
Server transport layer
Server process
Connection establishment
Send request
seq: 8001
ack: 15001
A P
Data
bytes: 8001-9000
Receive
Send request
seq: 9001
ack: 15001
A P
Data
bytes: 9001-10000
Receive
Send request
seq: 15001
ack: 10001
A
rwnd: 3000
Data
bytes: 15001-17000
seq: 10001
ack: 17001
A
rwnd:10000
Connection termination
Time
Time
Time
Time

Connection Close

- Either of the **two parties** involved in exchanging data (client or server) can close the connection
- It is usually **initiated** by the **client**.
- Most implementations today allow two options for connection termination:
  - **Three-way handshaking**
  - **Four-way handshaking** with a half-close option

Contd…

Three way Handshaking

Step-1

- The client TCP, after receiving a close command from the client process, sends the first segment,
- It is a FIN segment in which the **FIN** flag is set.
- A FIN segment can include the **last chunk of data** sent by the **client** or it can be just a control segment
- If it is only a control segment, it consumes only **one sequence** number because it needs to be **acknowledged**.

Contd…

Three way Handshaking

Step-2

- The server TCP, after receiving the FIN segment, informs its process of the situation and sends the second segment
- It is 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.
- This segment can also contain the **last chunk of data** from the **server**.
- If it does not carry data, it consumes only one sequence number because it needs to be acknowledged

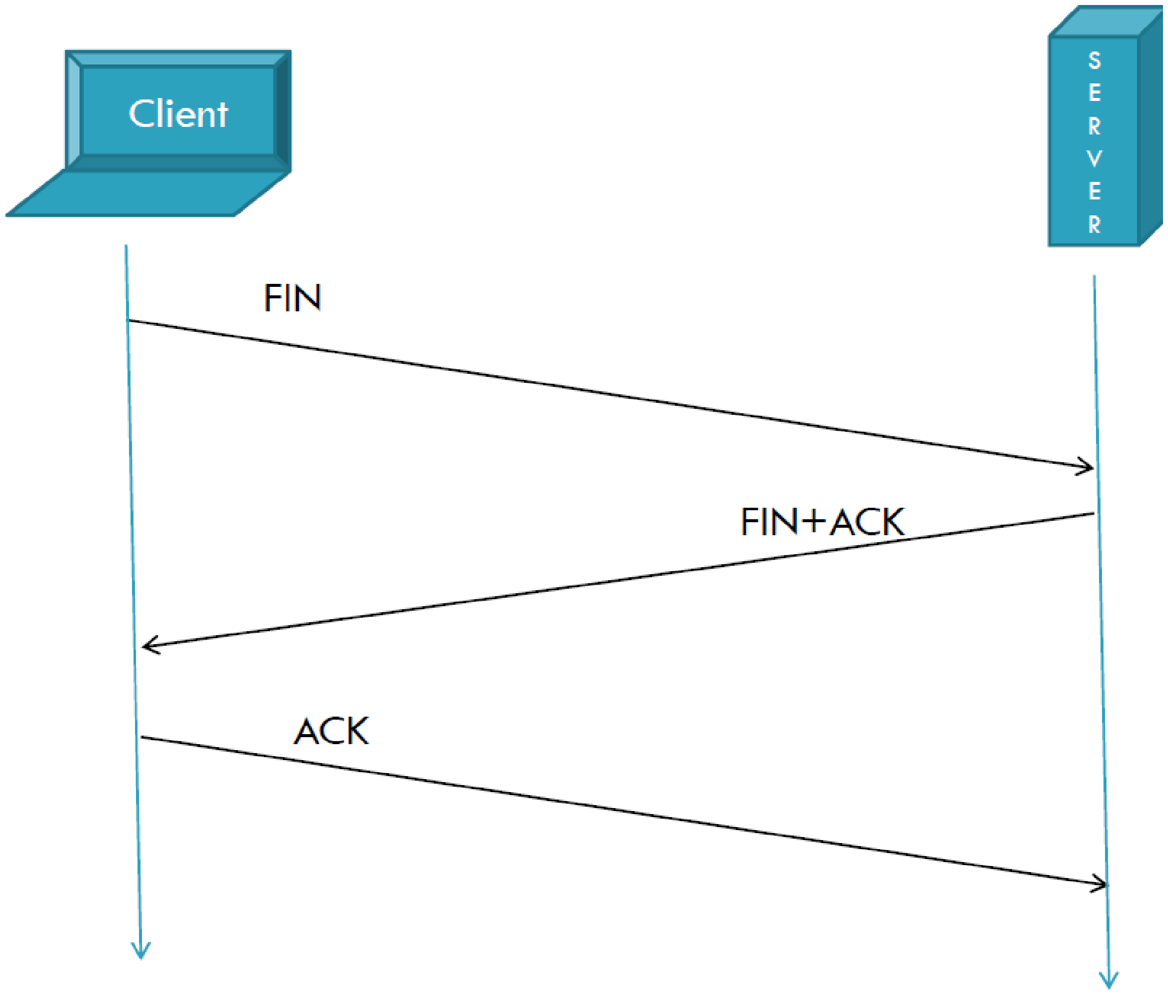
Contd…

Three way Handshaking

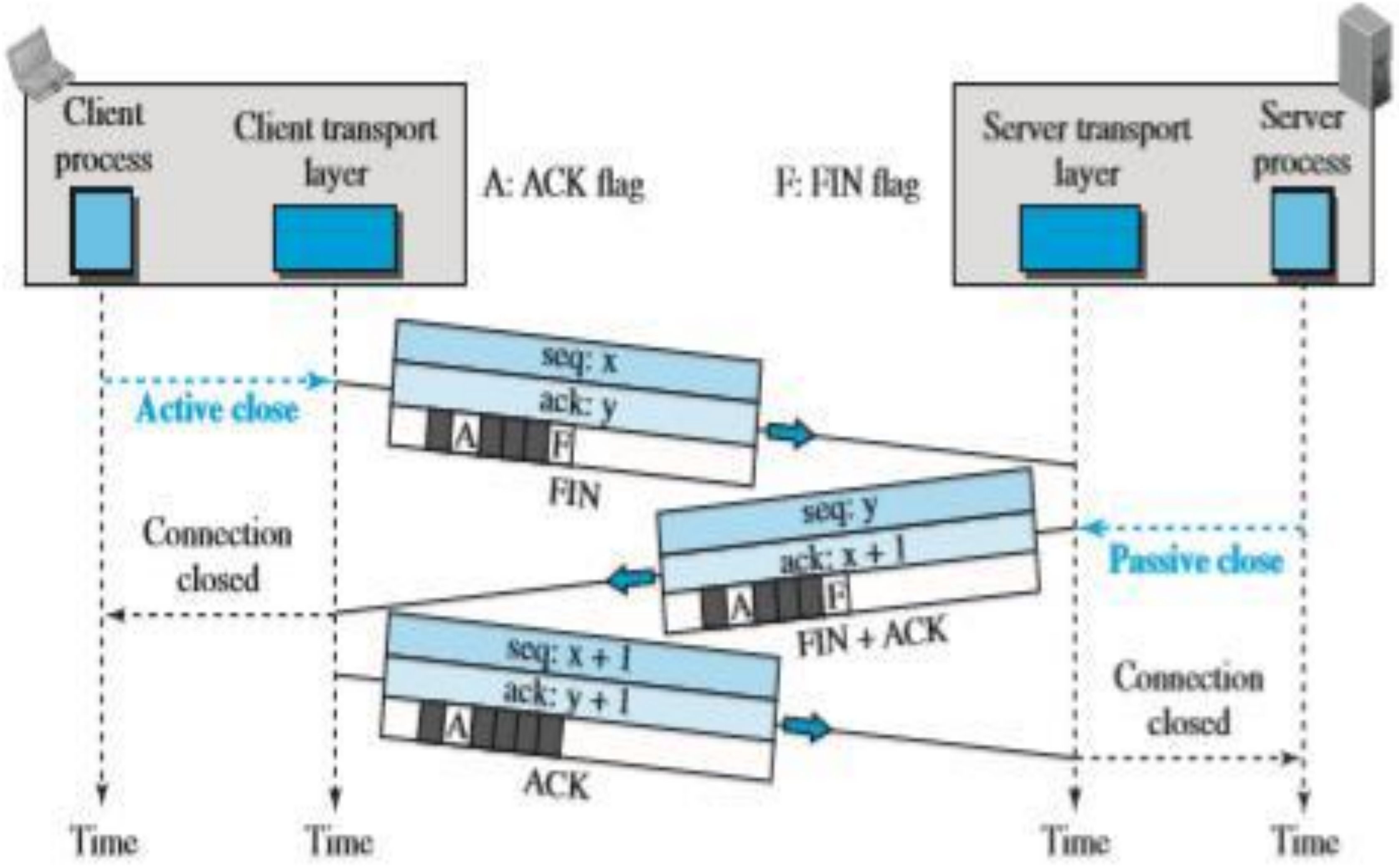
Step-3

- The client TCP sends the **last** segment
- Its an **ACK** segment, to confirm the receipt of the FIN segment from the TCP server.
- This segment contains the acknowledgment number, which is **one plus the sequence number** received in the FIN segment from the server.
- This segment **cannot carry data and consumes no sequence numbers**.

Steps of Three way Handshaking



Contd…



Client process
Client transport layer
A: ACK flag
F: FIN flag
Server transport layer
Server process
seq: x
ack: y
A
F
FIN
Active close
Connection closed
seq: y
ack: x + 1
A
F
FIN + ACK
Passive close
seq: x + 1
ack: y + 1
A
ACK
Connection closed
Time
Time
Time
Time

Connection Close

Half close

- In TCP, one end can **stop sending data while still receiving data**. This is called a half close.
- Either the **server or the client** can issue a half-close request.
- It can occur when the server needs all the data before processing can begin.
- After **half-closing** the connection, data can travel from the server to the client and acknowledgments can travel from the client to the server.
- The client cannot send any more data to the server

Contd…

Half close

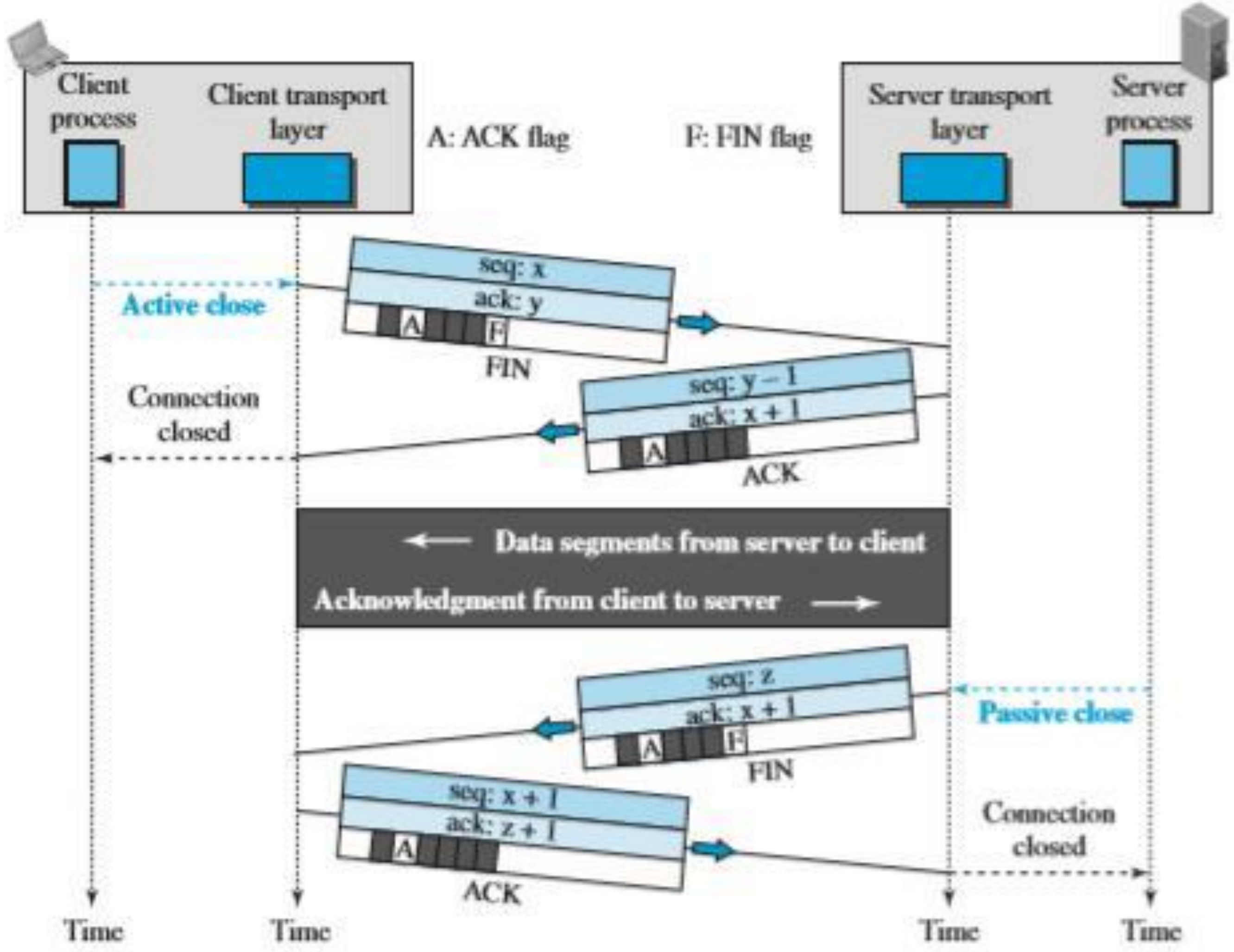
- **Example:** **Sorting**
- When the client sends data to the server to be sorted, the server needs to receive all the data before sorting can start.
- This means the **client**, after sending all data, **can close the connection** in the client-to-server direction.
- The server, after receiving the data, still needs time for sorting; its outbound direction must remain open.
- However, the server-to-client direction must remain open to return the **result i.e. the sorted data.**

Contd…

Half close

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

Contd…


Client process
Client transport layer
A: ACK flag
F: FIN flag
Server transport layer
Server process
seq: x
ack: y
A
F
Active close
FIN
seq: y – 1
ack: x + 1
A
Connection closed
ACK
Data segments from server to client
Acknowledgment from client to server
seq: z
ack: x + 1
A
F
Passive close
FIN
seq: x + 1
ack: z + 1
A
ACK
Connection closed
Time
Time
Time
Time

Connection Reset

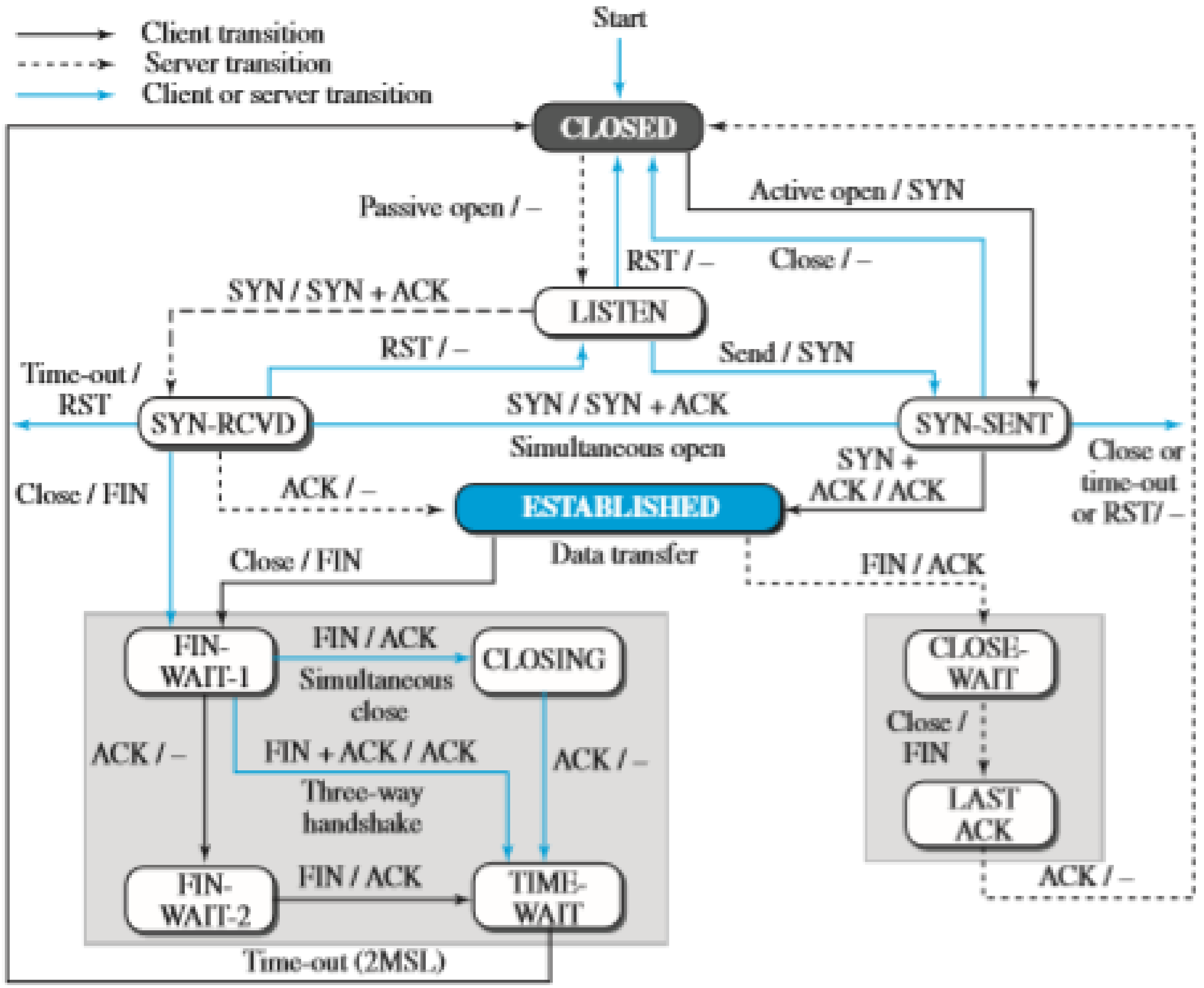
- TCP at one end may **deny** a connection request, may **abort** an existing connection, or may **terminate an idle** connection.
- All of these are done with the **RST** (reset) flag.

TCP states

- To keep track of all the different events happening during connection establishment, connection termination, and data transfer, TCP is specified as the **finite state machine**

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

Contd...


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

TCP state transition

- The rounded-corner rectangles represent the states.
- The transition from one state to another is shown using directed lines. Each line has two strings separated by a slash.
- The first string is the input, what TCP receives.
- The second is the output, what TCP sends.
- The dotted black lines represent the transition that a server normally goes through
- The solid black lines show the transitions that a client normally goes through.
- The colored lines show special situations.
- The rounded-corner rectangle marked ESTABLISHED is two sets of states: a set for the client and another for the server, that are used for flow and error control

Example of TCP state

Scenario: A half close scenario

Client side

- A client process issues an active open command to its TCP to request a connection to a specific socket address.
- TCP sends a SYN segment and moves to the SYN-SENT state.
- After receiving the SYN + ACK segment, TCP sends an ACK segment and goes to the ESTABLISHED state.
- Data are transferred, possibly in both directions, and acknowledged.
- When the client process has no more data to send, it issues a command called an active close.

Contd…

Scenario: A half close scenario

Client side

- The TCP sends a FIN segment and goes to the FIN-WAIT-1 state.
- When it receives the ACK segment, it goes to the FIN-WAIT-2 state. When the client receives a FIN segment, it sends an ACK segment and goes to the TIME-WAIT state.
- The client remains in this state for 2 MSL seconds
- When the corresponding timer expires, the client goes to the CLOSED state.

Contd…

Scenario: A half close scenario

Server side

- The server process issues a passive open command.
- The server TCP goes to the LISTEN state and remains there passively until it receives a SYN segment.
- The TCP then sends a SYN + ACK segment and goes to the SYN-RCVD state, waiting for the client to send an ACK segment.
- After receiving the ACK segment, TCP goes to the ESTABLISHED state, where data transfer can take place.
- TCP remains in this state until it receives a FIN segment from the client

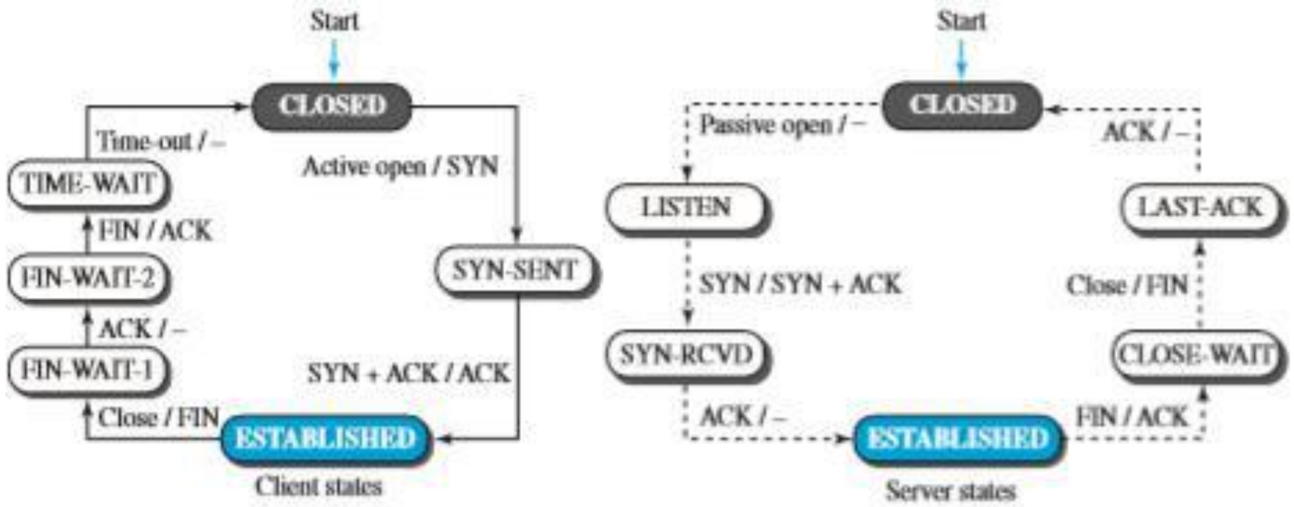
Contd…

Scenario: A half close scenario

Server side

- The server, upon receiving the FIN segment, sends all queued data to the server with a virtual EOF marker, which means that the connection must be closed.
- It sends an ACK segment and goes to the CLOSEWAIT state, but postpones acknowledging the FIN segment received from the client until it receives a passive close command from its process.
- After receiving the passive close command, the server sends a FIN segment to the client and goes to the LASTACK state, waiting for the final ACK.
- When the ACK segment is received from the client, the server goes to the CLOSE state

Example of TCP state



TCP Timers

To perform their operations smoothly, most TCP implementations use at least four timers:

- 1.retransmission
- 2.persistence
- 3. keep alive
- 4. TIME-WAIT

Retransmission Timer

Use: To retransmit lost segments

- TCP employs one retransmission timer (for the whole connection period) that handles :
  - The retransmission time-out (RTO),
  - The waiting time for an acknowledgment of a segment.
- To calculate the retransmission time-out (RTO), we first need to calculate the roundtrip time (RTT).

Contd…

RTT

- Measured RTT (RTTM): The measured RTT for a segment is the time required for the segment to reach the destination and be acknowledged
- Smoothed RTT (RTTs): It is a weighted average of RTTM and the previous RTTs
- RTT Deviation (RTTD): It is the calculate deviation, of RTT based on the RTTS and RTTM
- Retransmission Time-out (RTO): The value of RTO is based on the smoothed roundtrip time and its deviation
  - Original → Initial value
  - After any measurement → RTO = RTTS + 4 × RTTD

Persistence Timer

Use: To correct the deadlock between receiver TCP and sender TCP

- If the receiving TCP announces a window size of zero, the sending TCP stops transmitting segments until the receiving TCP sends an ACK segment announcing a non zero window size.
- If this acknowledgment is lost, the receiving TCP thinks that it has done its job and waits for the sending TCP to send more segments.
- There is no retransmission timer for a segment containing only an acknowledgment.
- The sending TCP has not received an acknowledgment and waits for the other TCP to send an acknowledgment advertising the size of the window.
- Both TCP's might continue to wait for each other forever (a deadlock).

Keep-Alive Timer

Use: to prevent a long idle connection between two TCPs.

- Suppose that a client opens a TCP connection to a server, transfers some data, and becomes silent.
- Perhaps the client has crashed.
- In this case, the connection remains open forever.
- To remedy this situation, most implementations equip a server with a keep alive timer.

Contd…

- Each time the server hears from a client, it resets this timer.
- The time-out is usually 2 hours.
- If the server does not hear from the client after 2 hours, it sends a **probe** segment.
- Probe 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.
- If there is no response after 10 probes, each of which is 75 seconds apart, it assumes that the client is down and terminates the connection.

Time-Wait Timer (2MSL)

Use: during connection termination.

- The **maximum segment lifetime (MSL)** is the amount of time any segment can exist in a network before being discarded.
- The implementation needs to choose a value for MSL.
- Common values are 30 seconds, 1 minute, or even 2 minutes.
- The 2MSL timer is used when TCP performs an **active close** and sends the **final ACK**.
- The connection must stay open for 2 MSL amount of time to allow TCP to resend the final ACK in case the ACK is lost.
- This requires that the **RTO timer** at the other end times out and new **FIN and ACK segments are resent**