Day-2

__/__/__

Transport Layer Role and Services

- → session establishment, management and Termination
- → segmentation of data
- → windowing: Addressing of upper layer applications using port numbers
- → sequencing → flow control → multiplexing → error detection
- → supports for Reliable and unreliable data delivery mechanisms through acknowledgm
- → Transport layer protocols are:

TCP (mapping Transport to Applications)

Port number – (16 bits) (0 - 65535)

port numbers are used to identify the type of service.

- → well known ports (0 to 1023)
- → Registered ports (1024 to 49151)
- → Dynamic ports / Private ports (49152 to 65535)

Now Dynamic ports

FTP - 21 (Trans) → 4
Telnet - 23
HTTP - 80
(App Lay)

TCP

DNS - 53

TFTP - 69
SNMP - 161
(App Lay)

UDP
(Trans Lay)

→ All the above port numbers in transport layer, we only think about how many services are used here.

all the protocols of application layer are used here called services.

TL Functions

APP Layer data

piece 1	piece 2	piec 3

UDP Datagram

Header piece 1

Header piece 2

UDP Header provides for:

Source and destination Ports

or

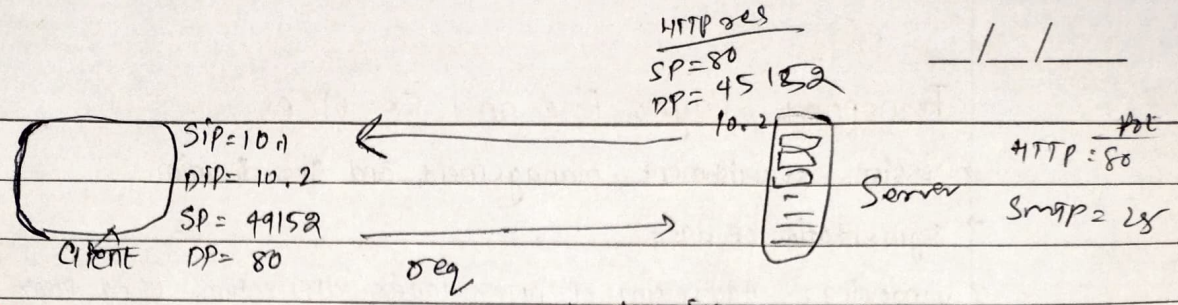
TCP segment

Header piece 2

Imp

TCP Header provides for:

- source and destination (Ports)
- sequencing for same order delivery
- Acknowledgment for received segment
- and congestion management



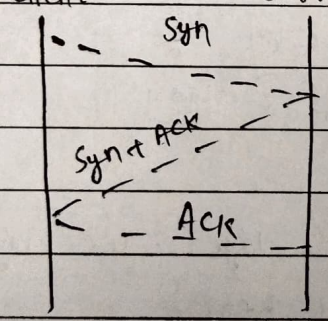
Transmission Control Protocol: (header size 20 bytes - 60 bytes)

imp MSS + SACK

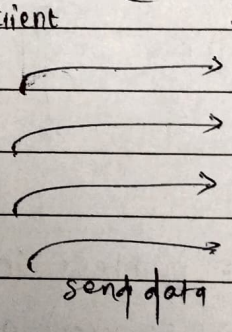
- RFC 793
- connection-oriented - Creating a session between source & destination
- Reliable delivery - Acknowledgement
- Order data Reconstruction - numbering and sequencing of segments
- Flow control - regulating the amount of data transmitted (windowing)
- error detection
- Stateful protection - keeping track of the session (Protocol e.g:- TCP)
- Tcp Applications: HTTP, SSH, SMTP

Establish a Connection

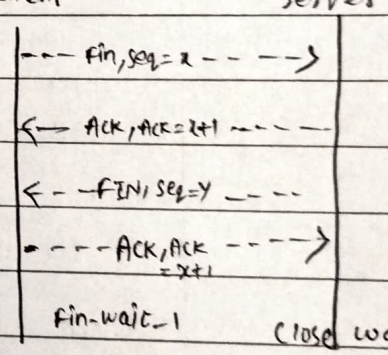
TCP (3 way)



UDP (faster)



TCP (4way)



Study - both side send FIN at the same Time

URGENT - for a particular segment

Push - empty the buffer

UDP (User Datagram protocol)

__/__/__

→ RFC 768

→ connectionless (→ No 3way) → unreliable (→ NO ACK) → No ordered data reconstruction (→ No seq)

→ No flow control (no windowing) → stateless protocol → UDP Application: (DNS, video streaming, voic over IP)

UDP header size

16-bit source port	16-bit destination port
16-bit UDP length	16bit UDP checksum
Data	

TCP segment and header:

Ethernet Header	IP Header	TCP Header	Payload	FCS
14 bytes	20 bytes	20 bytes	1460 bytes	4 bytes

TCP mess

IP MTU

ethernet MTU

MTU = maximum transmissin unit = 1500 + 14 (Ethernet Header) + 4 (FCS)

Payload/Data = 1500 − (TCP + IP) = 1500 − (20+20)
= 1460 byte

1460 + 20 + 25 (TCP) = 1505 (is known as Jumbo frame)

> 1500 (Fragmentation)

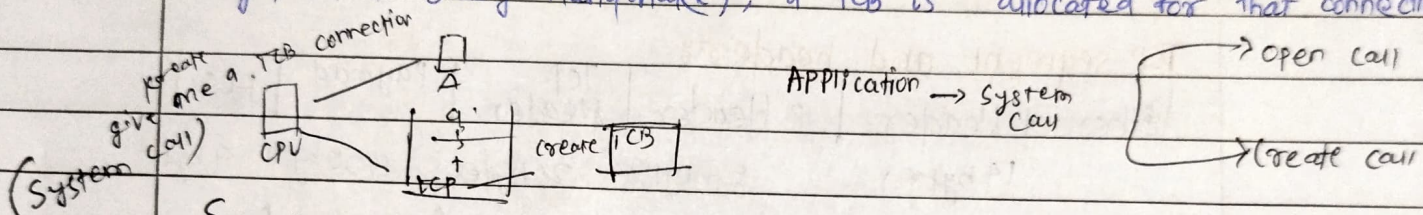
Tcp 3 way handshake:

(1) Why it happen? why do we need 3-way?

A: (i) establish a connection between source & destination
(ii) Initial proof of existance of destination side
(iii) Exchange of parameters MSS — maximum segment size
window size -

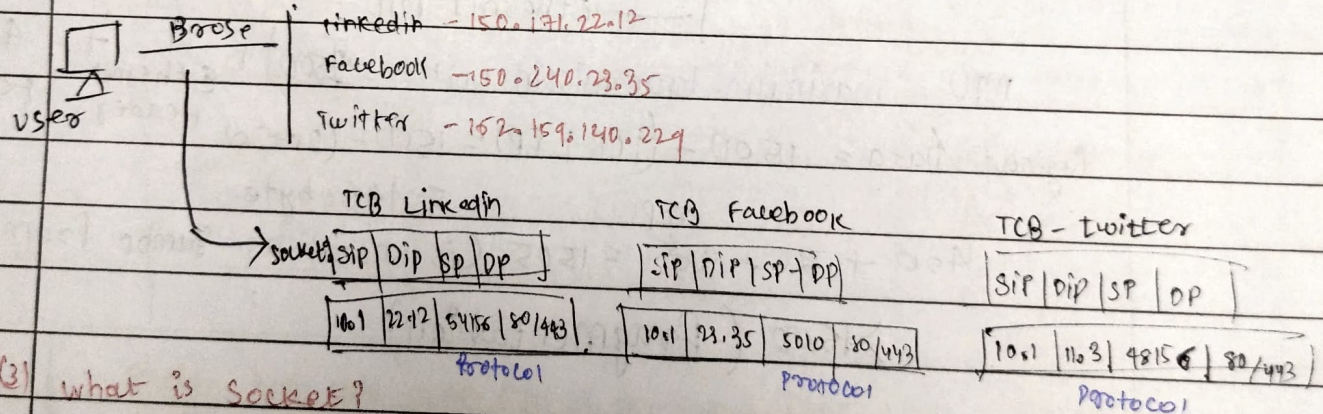
(2) What is TCB? (Transmission Control Block)

It is a data structure used by the TCP protocol to store information about each TCP connection. Everytime a TCP connection is created (e.g:, via a 3-way handshake), a TCB is allocated for that connection.



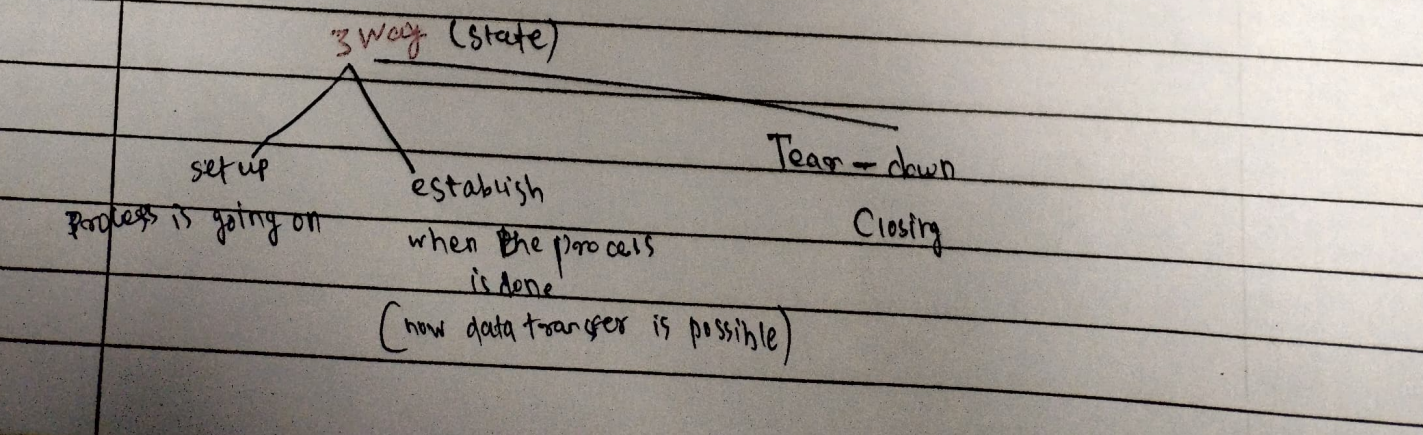
Socket

| SIP | DIP | SP | DP |



(3) What is Socket?

A: It acts as the communication end point used by application to send and Receive data over the network.



(4) How does the OS maintain uniqueness and lookup efficiently for TCP conn?

A: Yes, internally the operating systems uses a data structure like a map (or hash table) to manage TCBs, and these maps are keyed by socket identifiers to maintain unique port mappings.

Imp

TCB can be happened by 2 ways

Active - open

Passive open

Active - open: when CPU/TCP will create the TCB on demand

(5) How server Handles these many (100000 of) TCP requests at a time?

A: The server first creates a passive TCB (only local (server) info), and full active TCB (with SIP, DIP, SP, DP) is created only after receiving the client's SYN.

Stage	What TCB has	
listen()	Local IP (SIP) (server) + Local port (SP)	— Passive TCB, waits for c
After rece SYN	LIP + LP + Remote IP + RP	— Active TCB for this c

(6) How does a TCP server manage multiple clients using the same port?

A: A TCP server can handle multiple clients on the same port by using multiple sockets and either multi-threading or asynchronous I/O.

How it works??

Single port multiple clients

Client connected to server → Server assign unique source port + server

→ uniquely identified by the combination of server's IP and port (e.g.:- 192.168.1.1:8080) and client's IP and SP (e.g.:- 192.168.1.100:4

Connection handling

a new socket → a new client, but it listens for other clients on s (conn)

__/__/____

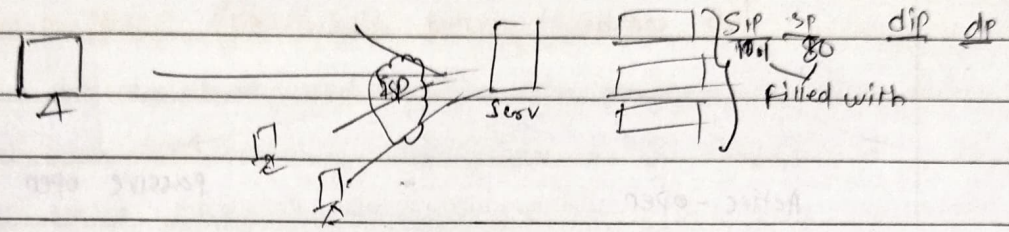
Handling multiple clients

- multi threading – a new thread for a connection each thread talks to one client
- Asynchronous I/O – uses event driven model (e.g: select, poll, or epoll on unix sys) to handle many clients at once without using multiple threads.

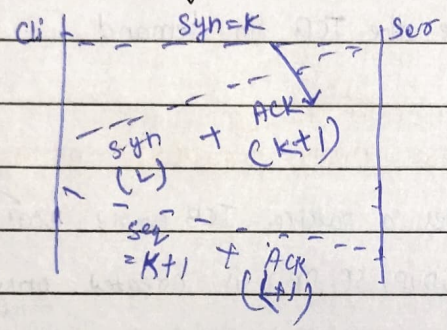
E.g:

web servers like APACHE or NGINX listen on port 80/443.
↳ Handles millions of clients at same time by using threading / Async.

Passive open:



ISN: (Identifying Sequence Number) (32 bit)



(Random number == Sequence number)

Q. How the sequence Number will increase??

A: It depends on segment Length – MSS (Max Segment Size)

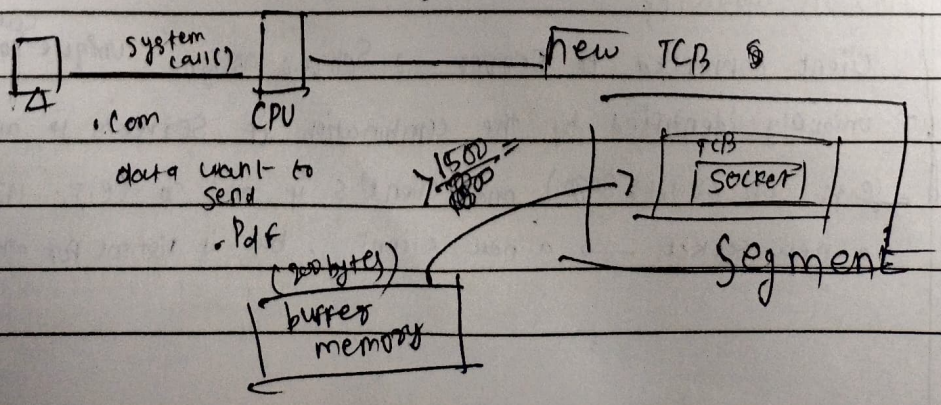
MTU = 1500 bytes

MSS = 1500 – (TCP + IP Header) = 1500 – 20 – 20 = 1460 bytes

↓ IPv4 = 20, IPv6 = 40 bytes

MTU (segment length) – 576 (min) — 576 – 40 = 536
– 1500 (max) — 1500 – 60/40 = 1460/1440

1518 = Eth header (14) + FCS (4)



Q. How many segments will be created??

File size > 1500 value/1500 + value/1500 = ~~6400/1500 = 4~~

value (6900) = 6900 % 1500 + 6900/1500 = 1 + 4 = 5

if 0 = 0
else 1

File in application layer (6900) byte → Transport layer

Seq=1 = 1500
seq=2 = 1500
seq=3 / seq=4 = 1500
seq=5 = 900

NOTE: MSS is inside TCP option.
↳ Byte in Flight

Day - 3

3way-handshake

i) Sq is increasing by 1 byte. (Phantom - Byte)

Q. What will be the next sequence number??

A: The sequence number depends on the previous sequence and the segment length (MSS).

Syn sq=1, segment length = 100

next seq will be = sq + segment length = 1+100 = 101

sq = 4742, Segment length = 10

4742	43	44	45	.	.	.	.	.	501
1	2	3	4	5	6	7	8	9	10

ACK = 4751+1 = 4752 next seq = 4750

Seq = 55

55	54	53	52	4951
5	4	3	2	1

Seq = 4755
Ack = 4755+1 = 4756

Segment length = 20 Byte

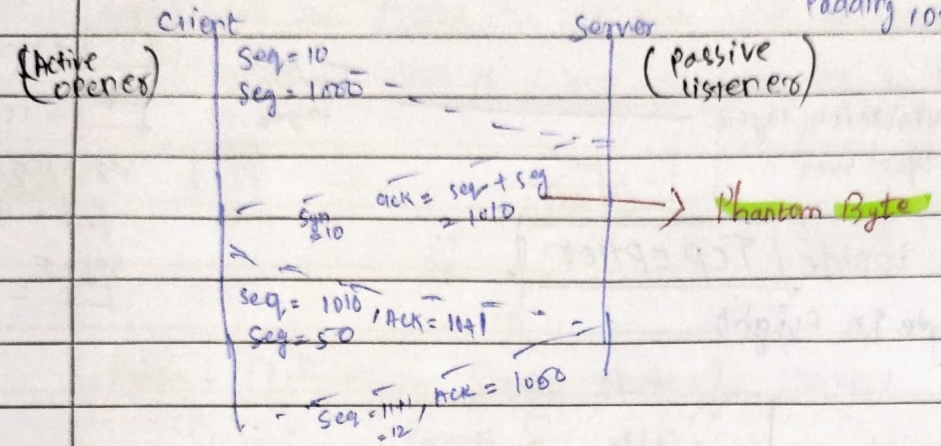
4775							57	56	4755
20									

Seq = 4775 ACK = 4775+1 = 4776

__/__/__

Q. what is phantom byte ??

A: A phantom byte is occurs in tcp when there is a mismatch or new sequence number because of misinterpretation, Software bugs, Padding or Protocol mismatches



Nagle Algorithm & GO

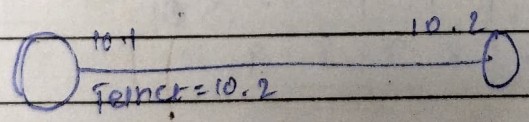
Negie's Algorithm is a method used by Tcp to reduce the number of small packets (called tinygrams) sent over the network, there by minimizing network congestion and improving efficient.

Eg: send 5bytes — it will send (step-1) ! before getting the first acknowledgment (step-2)
3bytes
2 bytes
it doesn't send 2nd packet rather it will wait for more data to come.

Q. why it is important?

A: message is 3 byte but (IP + TCP) header which is 60 byte. (Header > message)

NOTE:

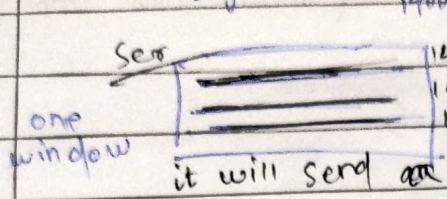


obviously it send smaller size message that is (3+10) Bytes of data each time. In this cases, the OS disables the "Nagle and enables the GO" algorithm.

Window: Without receiving any Ack how many segment i can send.
By default 16 bits = 2^{16} = 65535 bits but it can be increased upto 30 bi.

Suppose: Total data = 4168 MSS = 1400 if remainder = 0 'same' else +1

Total segment = $\frac{4168}{1400}$ = 2+1 = 3



After getting 4168 or more than that it will send an acknowledgment packet.

Q. How to increase the window size ?

A: using window SCALE concept

shift the bits to the left side.

8 =

1	0	0	0
3	2	1	0

8 4 2 1

16 =

1	0	0	0	0
4	3	2	1	0

16 8 4 2 1

Window - OS do
Change - bytes
MSS - manual

Day - 4 (SACK) (Sequence number)

Segment length = 10 bytes / bytes in flight

Seq = 101

Next Seq number = 101 + 10 = 111

ACK = 101 + 10 = 111, Am expecting seq. no - 111

Seq len = 5

Seq = 111

Next seq num = 111 + 5 = 116

ACK = 111 + 5 = 116, Am expecting seq. no - 116

Seq = 5

Seq = 116

N.S = 116 + 5 = 121

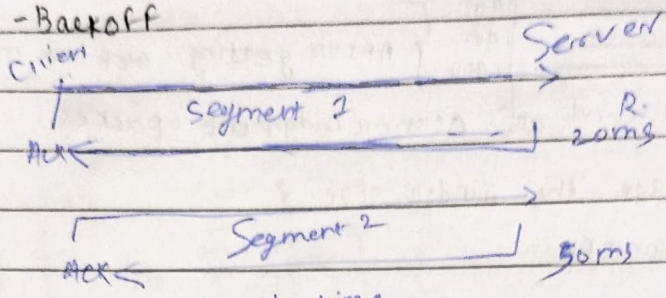
__/__/____

Timeout / Retransmission SRTT, IRTT, RTO

→ Retransmission Queue → IRTT (Initial Round Trip Time)

→ RTO (Retransmission Time out) → SRTT (Smooth Round Trip Time)

→ Time - Backoff

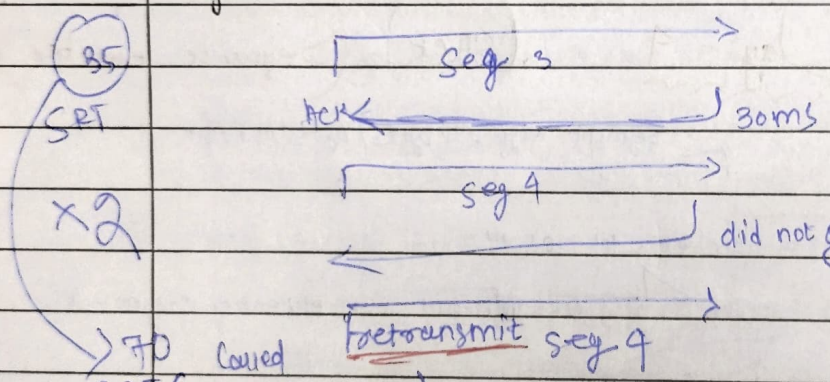


default guess for round-trip-time

IRTT: It is a method used to calculate average of round trip time.

SRT calculated First ms = 20 Second = 50 Avg = $\frac{20+50}{2}$ = 35 ms

If i am not getting any acknowledgment, it will send that packet again.



Called (Timer Backoff)

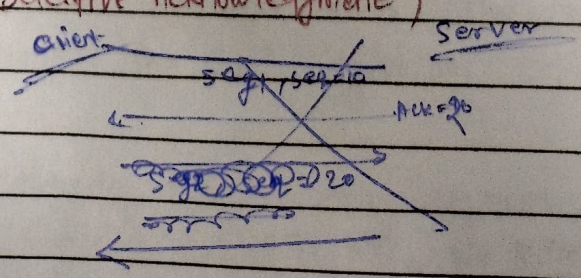
IRTT - Set initial RTO - Client - not

↓ changes while retransmission of same message

(Retransmission uses queue data structure to maintain RTO)

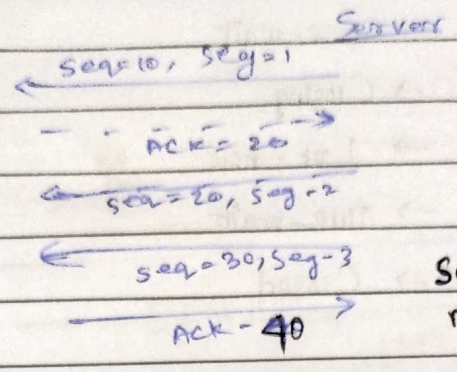
NOTE: During Retransmission the sequence number remaining same.

SACK: (Selective Acknowledgment)



without Selective Ack:

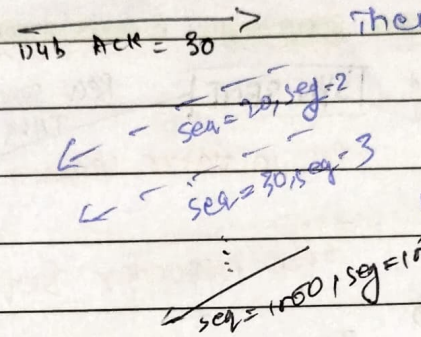
Total seg 1, 2, 3 ... 100



Server waits till RTO before sending missed acknowledgment (Segment) pack

what device does.

It send an duplicate acknowledgment packet



Then Server retransmits both seg-2 and seg-3

what servers does, it transmit all the segment ≥ 2 (2.... 1 (59:41)

With Selective Ack:

If the client send duplicate Acknowledgment then the send / retransmit only the specific / missed packet / segment.

NOTE: MSST SACK = present in Option

Day-5

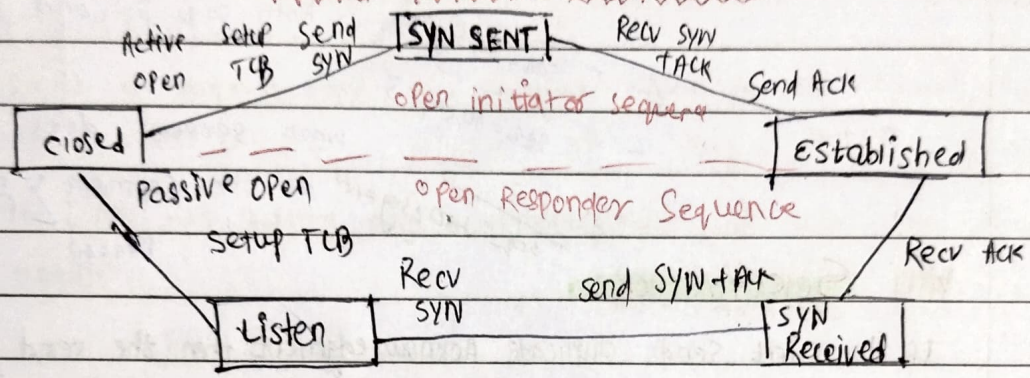
__/__/____

All TCP states

- → Listen
- → Syn-Sent
- → Syn-Received
- → established
- → Fin-Wait-1
- → Fin-Wait-2
- → Close-wait
- → Closing
- → Last-Ack
- → Time-wait
- → Closed

debugging
- State
- Transitions
- Event
- Action

Normal connection establishment



1st time

Client (Active opener) — Server (Passive opener)

(i) client —SYN→ Server

client (closed state – SYN SENT)

(ii) client (sync sent) ←sync+ACK— Server (closed – Listen state)

client (syn sent – established) ← (ii) client —ACK→ server (Listen – SYNC Received state)

Practice 2nd time

client (Active opener) — Server (Passive opener)

(1) closed — (1) closed

TCB — TCB's

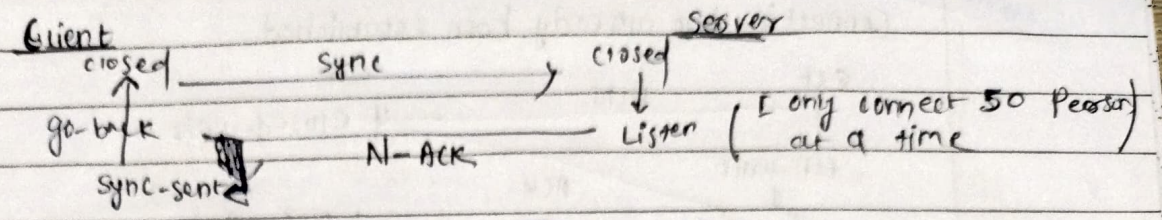
syn sent —SYN→ (2) Listen state

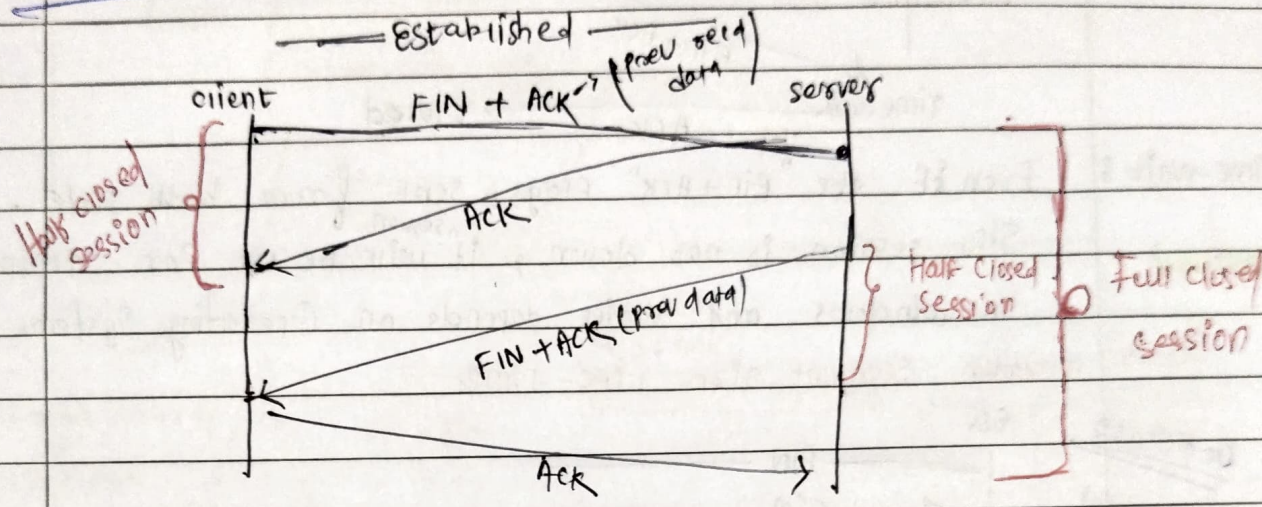
(2) SYN-SENT — (3) SYN-Received

SYN+ACK ←

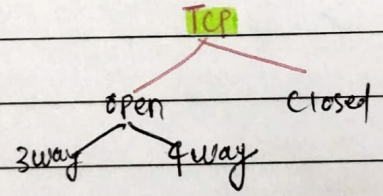
ACK →

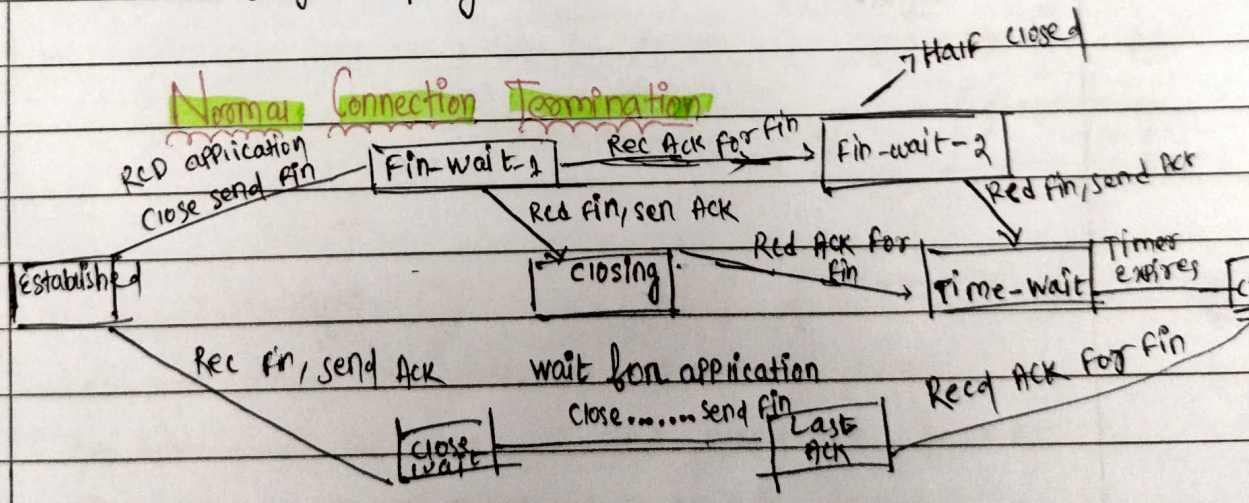
(3) Established — (4) Established





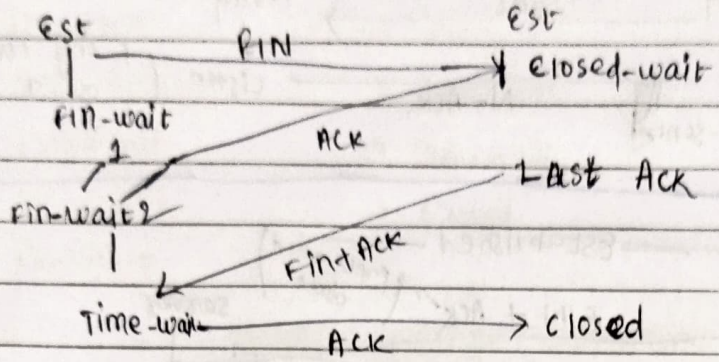
Half closed session: Termination of session from one-side





Condition -1

Connection has already been established



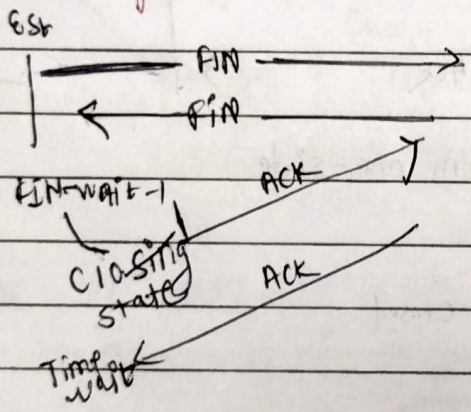
Time-wait : Even if the "FIN+ACK" flags is sent from both side. Still session is not down, it session will be up for 4minutes in windows and varies depends on Operating System.

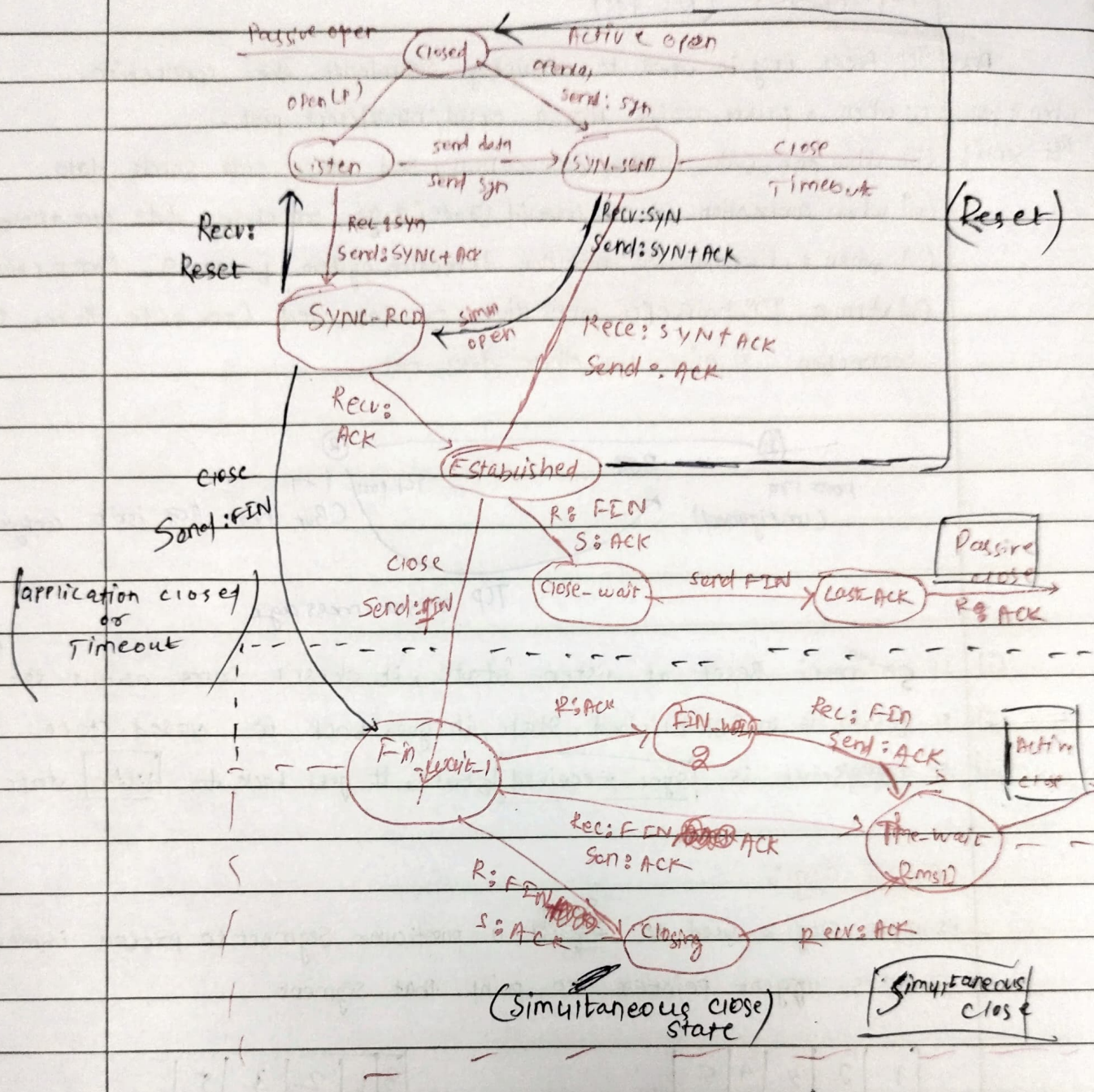
(MSSLT)

→ maximum Segment size Life-Time

Condition: 2

FIN (simu)
FIN





Q. SIMULTANEOUS OPEN STATE (occurs in BGP & P2P)

NOTE: (i) when both sends sync packet at the same time

(ii) Then both server and client will send SYN+ACK packet.

(iii) Then both server and client will send ACK packet.

(iv) Here the Server never goes to sync-received state, it stays in sync-sent state until it will get acknowledgment packet.

Tcp Reset (RFC 793)

Def: Tcp Reset flag is used to abruptly terminate the connection.

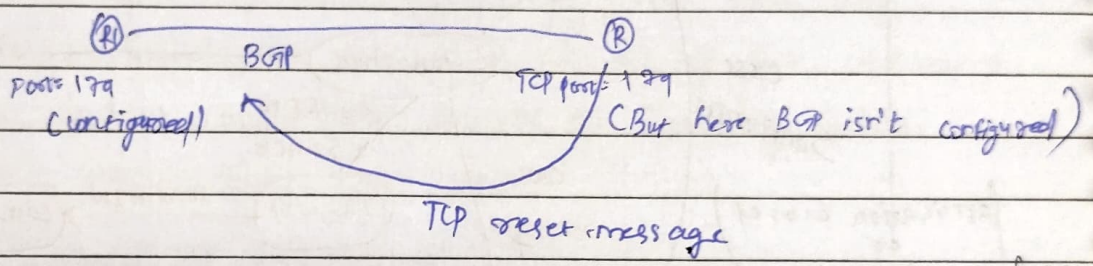
When it happens: (i) when a packet arrives for a closed/nonexistent port.

Rst sent?: (ii) when one side crashes or reboots, and other side sends data.

(iii) when connection is an invalid state (e.g:-, receiving data for a closed socket)

(iv) when a firewall or intrusion detection system blocks a connection.

(v) when a TCP half-open connection is detected (one side thinks the connection is alive, but other does not.



(i) If server recei Reset at listen state, it doesn't care about the flag.

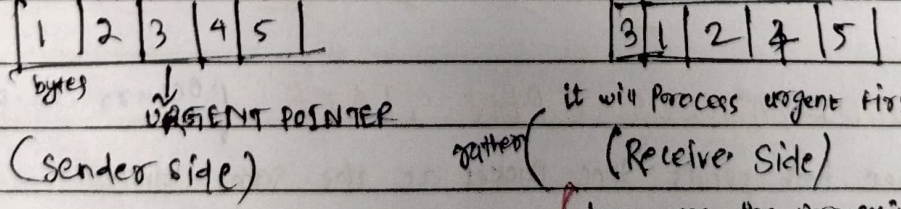
(ii) If server is in established state, it goes back to closed state

(iii) If client/Server is [sync_received] state, it goes back to [listen] state.

URGENT Flag:

→ URGENT Flag is used to ~~request~~ Send a particular segment to process immediately.

→ It uses urgent pointer to point that segment



Use cases:

When a user sends an interrupt command (like ctrl + c to stop a running process), the urgent pointer flags it for immediate processing.

Push Flag:

Send whole buffer memory data insted of waiting to load whole data.

(NONCE, ECN, ECN-echo, CWR) congestion control, TCP OPTIONS