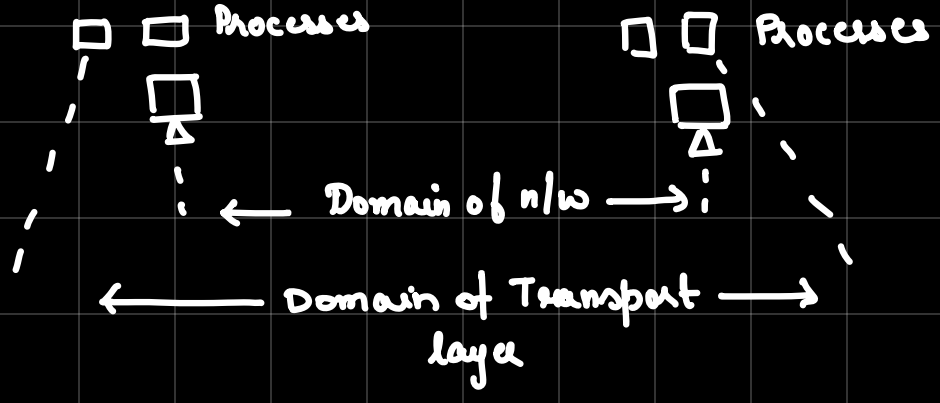
Transport Layer Services :

1. Process to Process communication



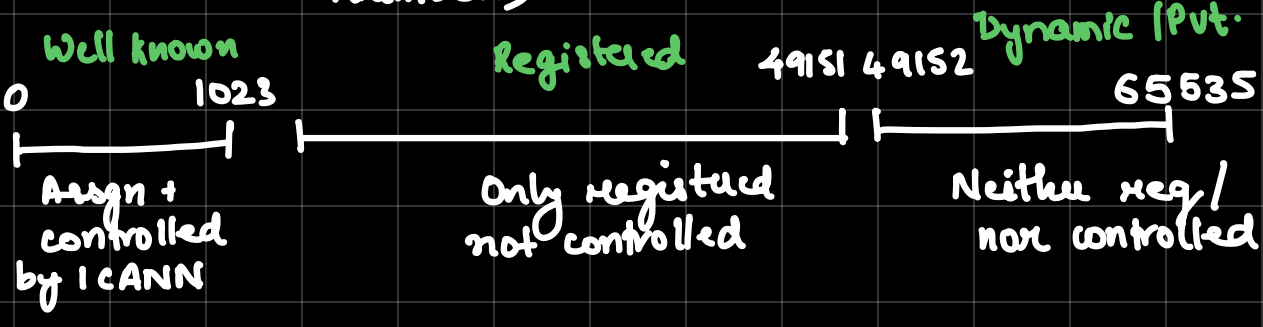
2. Addressing - to define processes we req. identifiers called port numbers.

↳ 16 bit add.

> Client defines itself with a port number called emphesal port number - short lived.

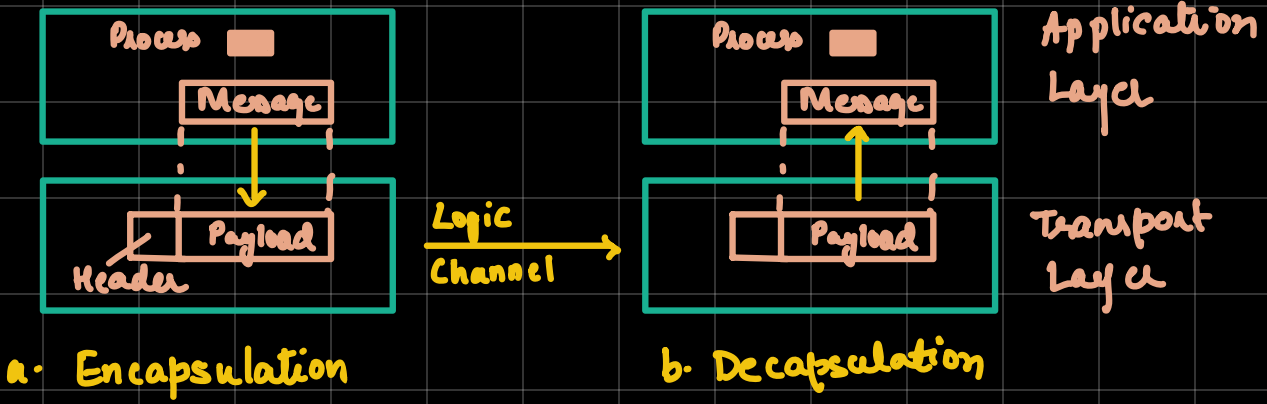
> TCP/IP has decided to use universal port numbers for servers - well known port numbers.

3. ICANN (Internet Corporation for Assigned Names & Numbers)



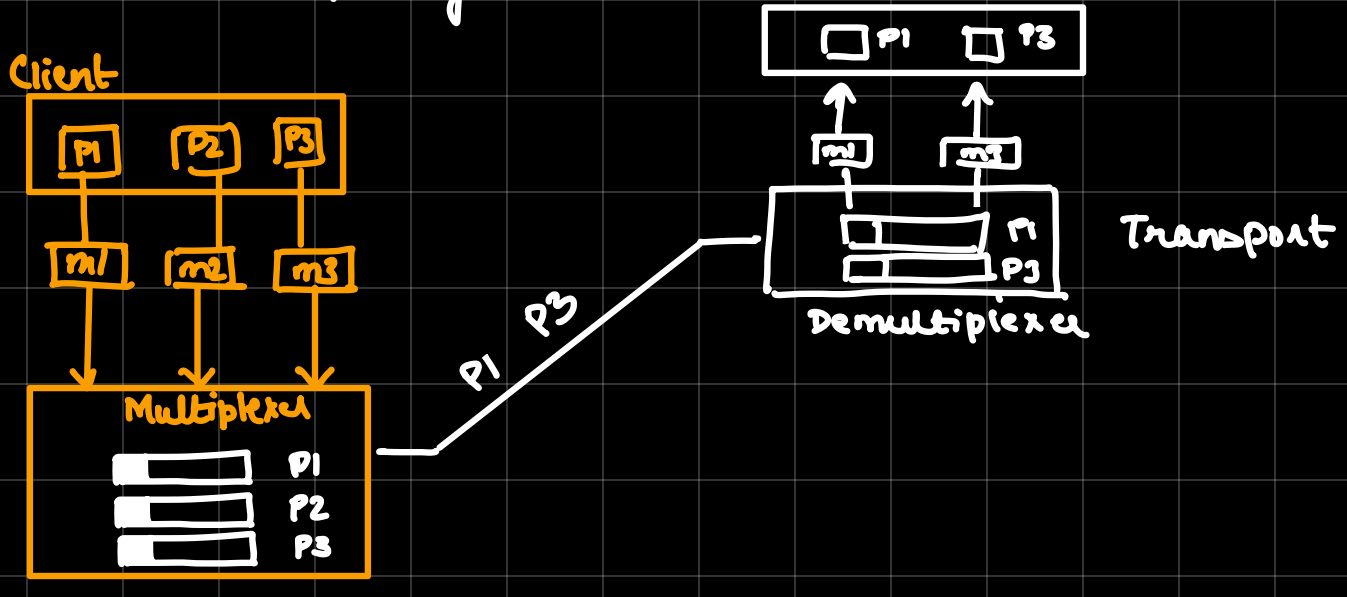
4. Encapsulation & Decapsulation

↳ adds the transport layer header to the message from the appn. layer on client side & vice versa on server side



5. Multiplexing & Demultiplexing

↳ the transport layer at the client side will perform multiplexing & at the server side it performs demultiplexing

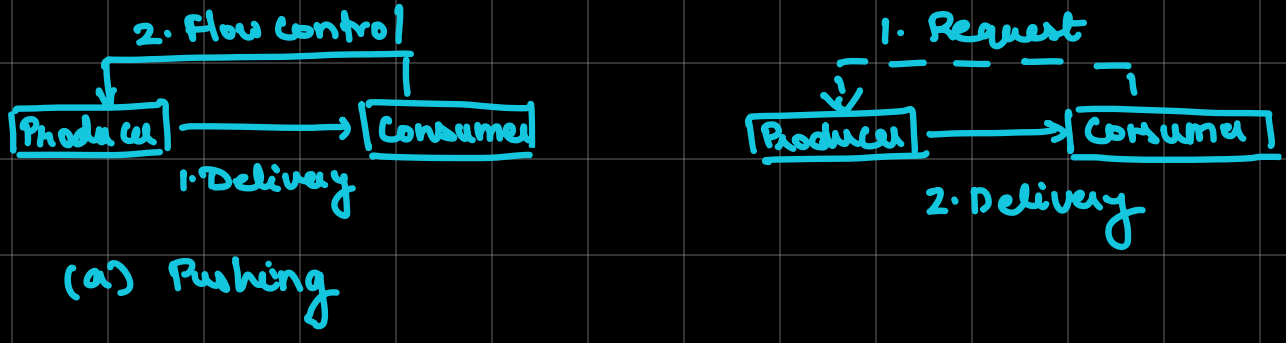


5. Flow Control

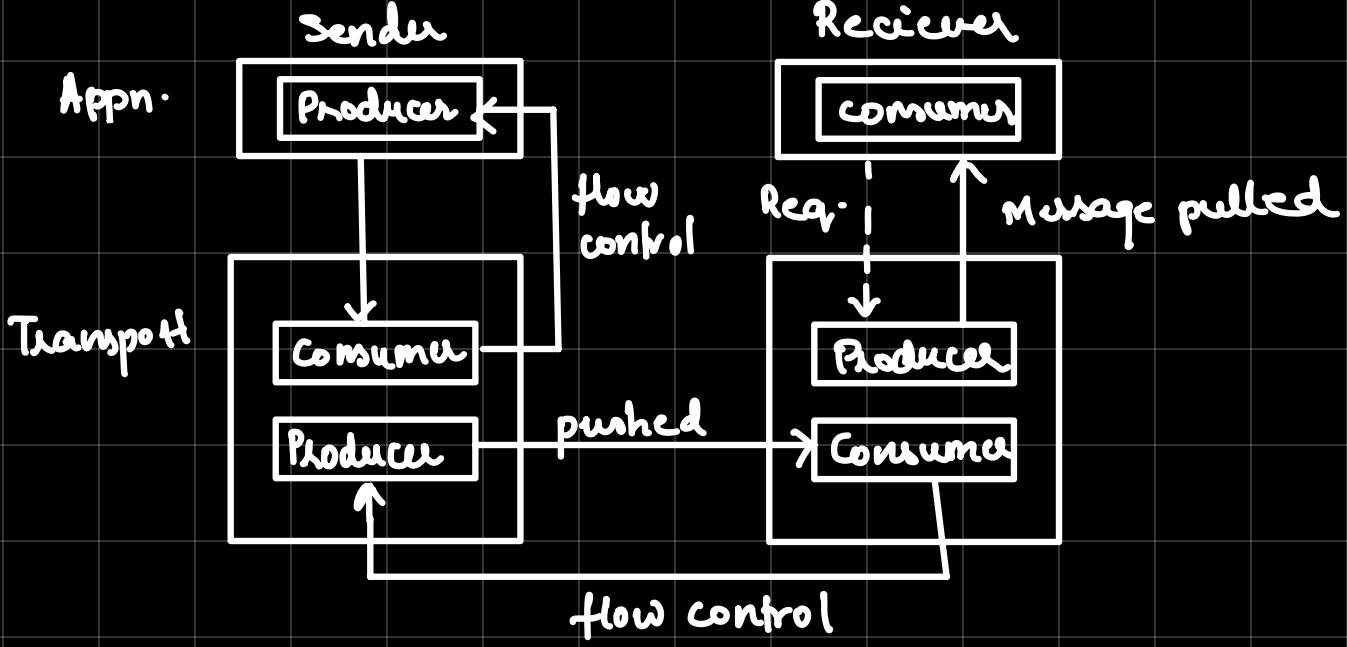
↳ balance b/w production & consumption rates

Pushing : if the sender delivers items whenever they are produced without a prior request from the consumer.

Pulling: if the producer delivers the items after the consumer has requested them.



(a) Pushing



Buffers → use 2 buffers : one at sending transport layer
the other at recieving "

> sending → full → stop passing chunks of mssg. to appn layer
> recieving → full → stop sending packets to sending transport layer

6. Error Control

1. Detecting & discarding corrupted packages
2. Keeping track of lost & discarded packages & resending them
3. Duplicated packages
4. Buffering out of order packets until missing packets arrive

7. Sequence numbers - modulo 2m ← size of sequence number field in bits

8. Acknowledgement - both positive & negative ACK signals as error control.

9. Congestion Control - refers to the mechanisms & techniques to control the congestion &

Keep the load below the capacity

↳ congestion in transport layer ⇒ result of congestion at n/w layer.

Finite State Machine:

→ behaviour of both connectionless and connection-oriented protocol can be better shown as a FSM.

→ each transport layer is thought of as a machine with finite no. of states.

*Diagram from notes

User Datagram Protocol

→ unreliable, connectionless transport layer protocol used for its simplicity & efficiency in applications where error-control can be provided by appn. layer process.

↳ why use?
- minimum overhead
- small messages, no need of reliability

7 - Echo
13 - Daytime
9 - Discard
11 - Users
53 - DNS

Header = 8 bytes

0	16	31
Source Port Number	Destination Port Number	
Total Length	Checksum	

Services:-

1) Process to Process Communication : using socket add. which is a combination of IP add + port no.

2) Checksum → Pseudo header - demo header for checksum on TCP/UDP packets. From TCP pov IP add. not included so to do a proper checksum

3) Connectionless service

4) Each datagram sent is independent (no rel^n. b/w datagrams even if same src → destn.)

5) Flow control ✗

6) Error control ✗ - no error control except checksum ↙ if error silently discarded

7) Congestion control ✗

8) Encapsulation & Decapsulation

9) Multiplexing & Demultiplexing

Appn:
1. Trivial File Transfer protocol (TFTP) has error + flow control → : UDP
2. Multicasting
3. Management process like SNMP
4. Route updating protocols like RIP
5. Real time appn. that cannot tolerate delay b/w sections of rec. msg

Transmission Control Protocol

Services:

1) TCP is a connection-oriented, reliable protocol.

↓ ↗ logical

Connection established → data transfer → connection termination

2) GBN + SR ⇒ provide reliability
   (GBN: go back N; SR: selective repeat)

3) Checksum (error detection), retransmission of lost or corrupted packages, cumulative/selective ACK, timers.

4) Process to Process Communication using port no.s

5) Stream Delivery Service: producer writes to the stream & consumer reads from it.

6) Segments: → n/w layer, as a provider to TCP needs to send data in pkts not stream of bytes
   → TCP groups no. of bytes into a packet called segment ⇒ Header added to every segment

7) Full Duplex Communication ie client & server can communicate simultaneously.

8) Multiplexing & Demultiplexing

First segment = ISN

9) Numbering System ← seq no. / ACK } byte no.

any other seq = prev seq + no. of bytes ↳ random

no. of the next byte ⇒ cumulative

octets - data bytes

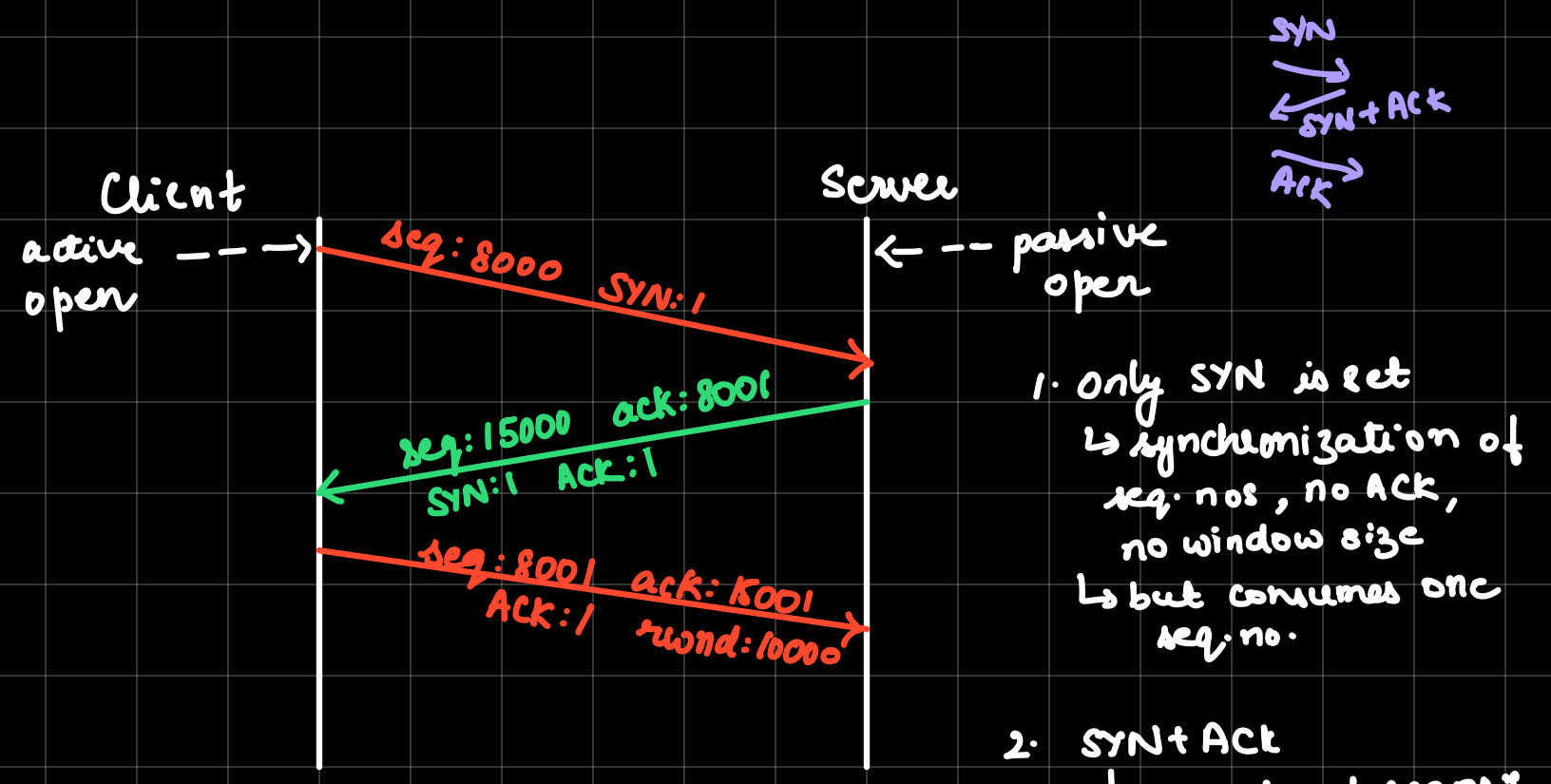
* TCP connection

→ Connection Establishment

- full duplex mode
- three-way handshake

server tells TCP it is ready to accept connection → PASSIVE OPEN

client issues a req. for an → ACTIVE OPEN



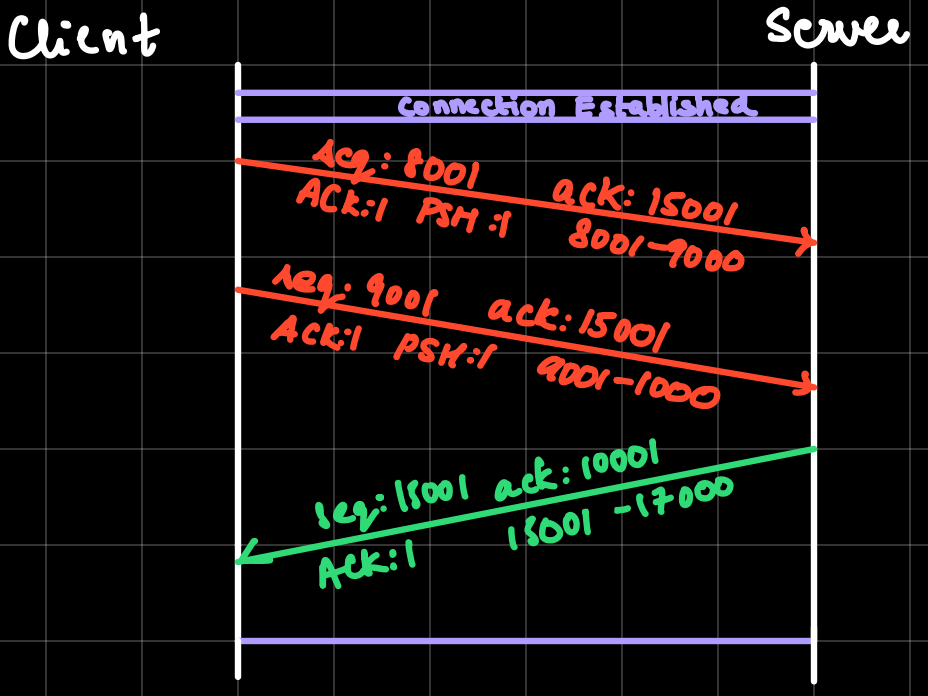
1. Only SYN is set
   ↳ synchronization of seq. nos, no ACK, no window size
   ↳ but consumes one seq. no.

2. SYN + ACK
   ↳ synch. of seq no.s server → client
   ↳ also ACK for SYN sent by client by setting SW flag
   ↳ NO DATA

3. Client send
   ↳ ACK but also sets window size
   NO DATA / NO SEQ consumed

→ Data Transfer

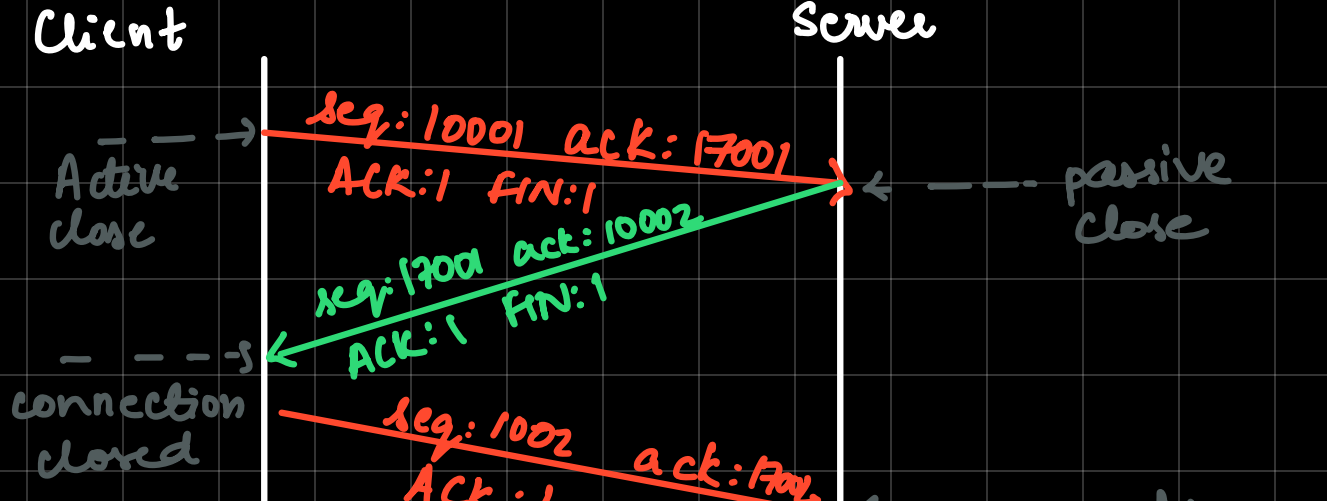
PSH → push



1. Piggybacking ACK + data
2. Pushing data
3. Urgent data
   URG:1 so this tells the urgent data to the beg.

→ Connection Termination

- three way handshaking
- or four way " with half close option



① Client TCP sends FIN with FIN flag set
↳ can include data ⇒ consumes seq no. if no data

② Server TCP after receiving FIN sends FIN+ACK
to announce closing on other direction
↳ can also carry data ⇒ if not consumes a seq. no.

③ Client TCP sends last segment an ACK segment
to confirm FIN + ACK
↳ does not consume seq. no

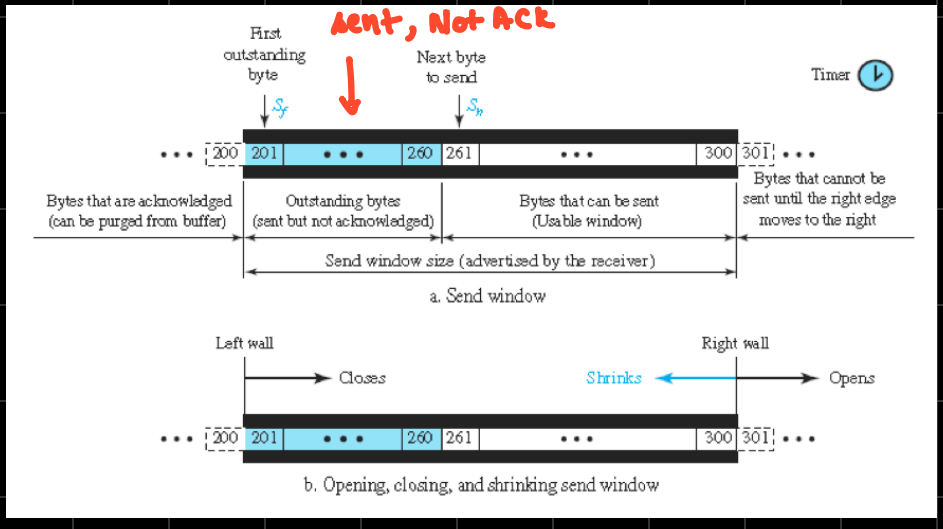
* Half Close

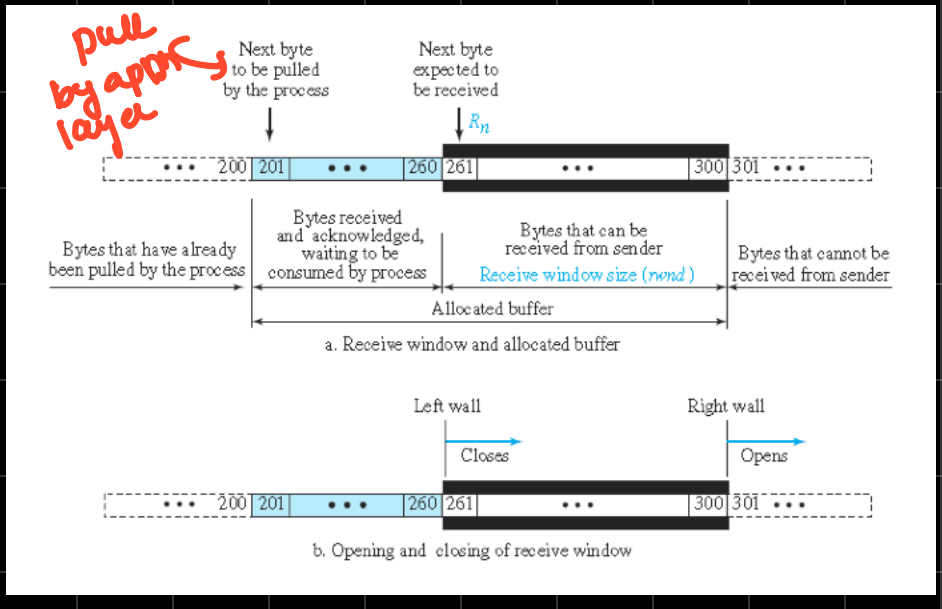
Connection Reset :

TCP at one end may : ① deny conn. req.
② abort existing conn.
③ terminate idle conn.

Windows in TCP :

Send window - size dictated by reciever (flow ctrl) & congestion in underlying nlw (cong. ctrl)





Diff b/w TCP & SR send window (note)

TCP header → 20 – 40 bytes

Source Port Address	Destination Port Address
Sequence Number	
Acknowledgement Number	
HLEN | | | | | | | | | |	Window size
Checksum	Window size
Options & Padding	

Flow Control

→ to achieve flow control, TCP forces sender & receiver to adjust their window sizes.

→ size of buffer of both parties ⇒ fixed at conn. establish

→ Opening, Closing & Shrinking of sender window controlled by reciever

SW left wall → closing (ACK allows)

right wall → receiver (rwnd) allows

RW left wall → moves bytes from sender pushed

right wall → opens bytes pulled.

* RW cannot shrink → SW can if rwnd results in shrinking
↳ some don't allow this aswell

↳ needs to keep relv. to prevent shrinking

new Ack + new rwnd ≥ last Ack + last rwnd

Congestion Control

- ↳ slow start
- ↳ congestion avoidance
- ↳ Fast recovery

→ TCP uses diff policies for congestion control

→ size of send window is controlled by the receiver using value rwnd

→ receive window is never overflowed with rec. bytes ↗ no end congestion

Congestion Window:

↳ congestion occurs in the middle ⇒ intermediate buffers of routers get congested. ↗ they receive data from many senders

↳ TCP needs to worry about congestion cause if it occurs ⇒ collapse comm.

↳ TCP defines policies that accelerate data transfer
⇒ no congestion & decelerate ⇒ congestion detected

cwnd ← congestion window → size controlled by congestion of n/w

∴ Actual send window = min (rwnd, cwnd)

Congestion Detection — 2 events

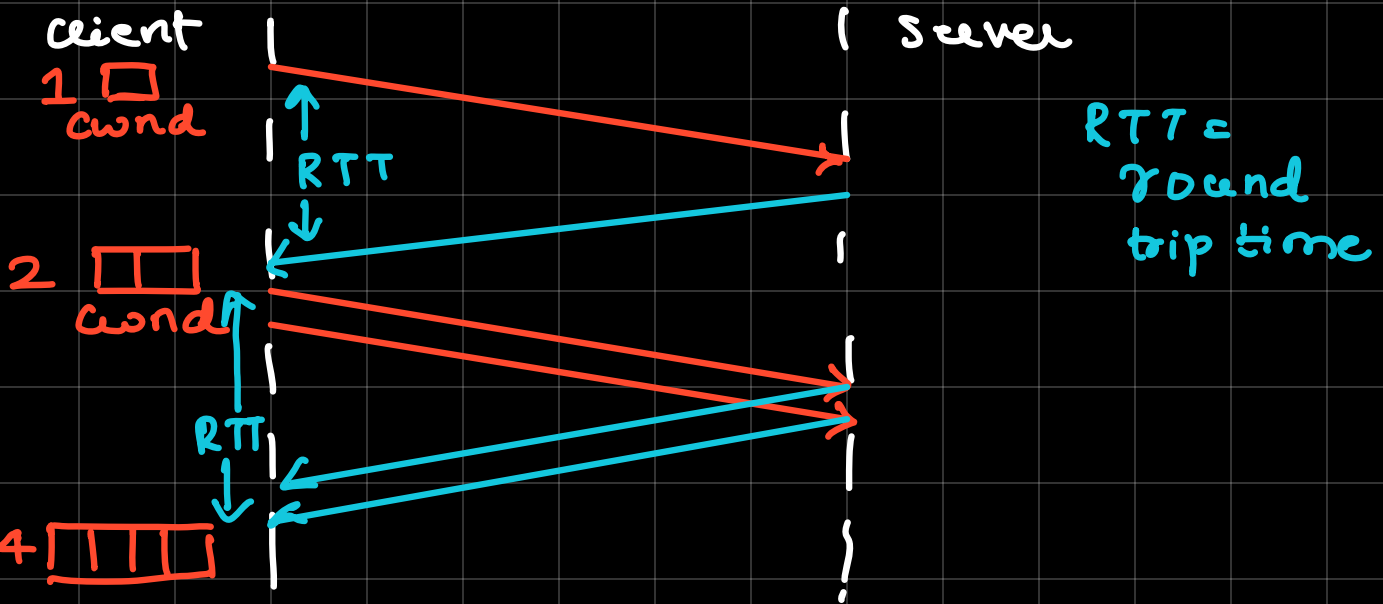
→ **Time out event** - sender doesn't receive ACK or segment btr time out occurs

→ **Receiving 3 dupe ACKs** -
- 1 duplicate Ack → delay of seg.
- 3 dup. Acks → congestion in n/w
- 3 Acks congestion < severe than time out

1. Slow Start (Exponential Increase)

→ • Assume that cwnd starts with 1 MSS (max segment size), but increase by 1 MSS each time an ACK arrives.

• rwnd >> cwnd ∴ sw = cwnd



Start → cwnd = $1 \rightarrow 2^0$

After 1 RTT → cwnd = cwnd + 1 = 1 + 1 = 2 (2^1)

2 RTT → = 2 + 2 = 4 (2^2)

3 RTT → = 4 + 4 = 8 (2^3)

ssthresh (slow-start threshold): size cwnd ↑ exponentially until it reaches threshold

If 2 segments ACKed cumulatively cwnd increases by 1 not 2

↳ 1 ACK / two segments power of = 1.5 ✓ 2 ✗

2. Congestion Avoidance - Additive Increase

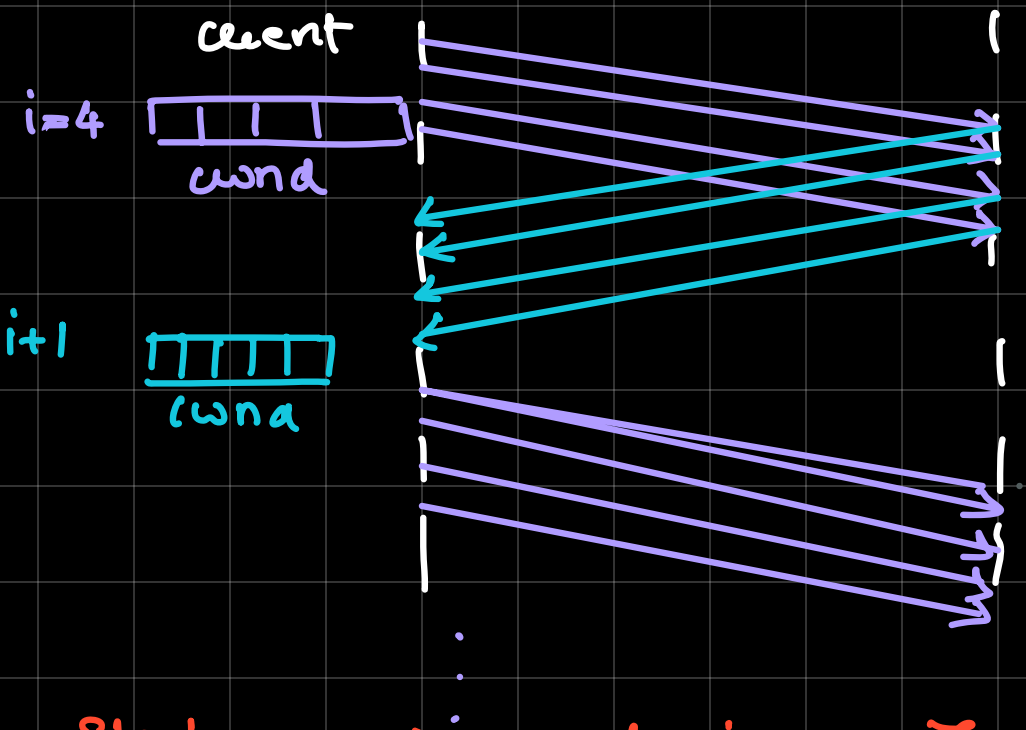
→ size of window reaches ssthreshold,

cwnd = i,

slow start stops & additive inc. begins

→ Now each time window of segments is ACK

size of cwnd, +1.



Start → cwnd = i
1 RTT → cwnd = i+1
2 RTT → cwnd = i+2

} +1 when congestion detected

3. Multiplicative Decrease

1. Time out occurs : segment has been dropped in the n/w → no news

TCP → strong reaction

a. sets value of threshold to half of current window size.
b. reduces cwnd back to one segment
c. starts slow phase again

2. 3 duplicate ACKs : a segment may have been dropped but some " after that have arrived safely - Fast Recovery.
↳ Optional in TCP

TCP → weaker reaction
a. set value of threshold to half of current window size
b. cwnd = threshold value
c. starts from congestion avoidance

TCP Timers

1. Retransmission 2. Persistence 3. KeepAlive
4. Time Wait

⇒ Retransmission

→ To retransmit lost segments, TCP employs one retransmission timer (for whole connection period)
→ handles RTO, the waiting time for an ACK of a segment.
→ TCP sends segment in front of standing

queue & starts timer (oldest segment sent out)

→ Queue empty ⇒ stop timer

queue = all sent segments not ACKed

⇒ Persistence Timer

⇒ to deal with zero-window size ad, TCP needs another timer.

→ Recieving TCP announces zero window size, so sending TCP stops segment transmission till non-zero ACK.

→ Both TCPs might continue to wait for each other ⇒ deadlock

→ So when a zero window ACK recieved it starts a "persistence timer" & when it goes off a probe sent (1byte data)

↳ causes recieving TCP to resend ACK

value of PT = retransmission time

↳ otherwise another probe, PT doubled

↳ (threshold = 60s)

↳ after that every 60s

⇒ Keepalive Timer

→ used to prevent long idle connection b/w 2 TCP

→ server resets timer each time it hears from the client

→ timeout ~ 2 hrs ⇒ send probe

→ no response after 10 probes (75s apart) ⇒ assumes client is down & terminates conn.

⇒ Time-Wait Timer (2MSL)

→ used during connection termination

→ Common MSL values → 30s, 1min, 2min

→ 2MSL is used when TCP performs active TCP, & sends final ACK.

→ conn. must stay open for 2MSL amt of time to allow TCP to resend ACK if lost.

* MSL → maximum segment life is the amt of time any segment can exist in a n/w before being discarded

1:2 notes !

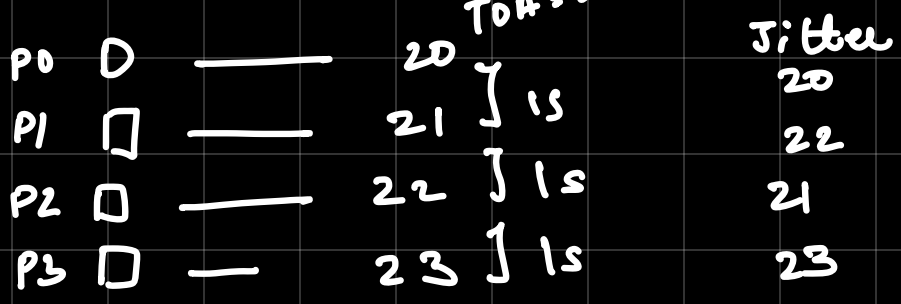
Quality of Service

Flow Characteristics
- Reliability
- Delay
- Jitter
- Bandwidth

Reliability: Lack of reliability means losing a packet or acknowledgement, which entails retransmission
- email reliability > audio conferencing

Delay: source → destination delay
diff apps can tolerate delays to diff degrees
- delay in email / file transfer < audio, video

Jitter: variation in delay of packets belonging to the same flow



* audio / video req. uniform delay blw packets

Bandwith: capacity of channel / transmission (mainly capacity)
↳ diff appn. require diff bandwiths

Techniques to improve QoS
- Scheduling
- Traffic Shaping
- Resource Reservation
- Admission Control

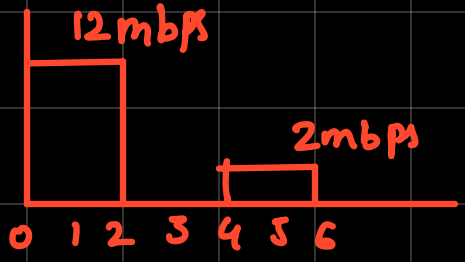
→ Admission Control – mechanism used by routers, switch to accept or reject flow based on predefined parameters called flow specifications

→ Resource Reservation – flow of data req. sources such as buffer, bandwith, CPU and so on. QoS can be improved if these resources are reserved.

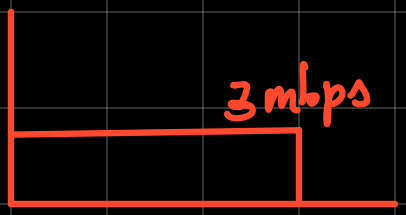
→ Traffic shaping – mechanism to control amt & rate of traffic sent to n/w

1. Leaky Bucket 2. Token Bucket

1. Leaky Bucket : algo shapes bursty traffic into fixed-rate traffic by averaging data rate. ⇒ drop packets if full



bursty



fixed

2. Token bucket : algo allows output rate to vary depending on size of burst, allows bursty traffic at regulated max rate.

TOKEN BUCKET	LEAKY BUCKET
Token dependent.	Token independent.
If bucket is full token are discarded, but not the packet.	If bucket is full packet or data is discarded.
Packets can only transmitted when there are enough token	Packets are transmitted continuously.
It allows large bursts to be sent faster rate after that constant rate	It sends the packet at constant rate
It saves token to send large bursts.	It does not save token.

→ Scheduling — FIFO
→ Priority Queue
→ Weighted "

↳ see diagrams from notes