GATE CSE NOTES

by

UseMyNotes

TCP, UDP

* Transport layer protocols :

1. Transmission control protocol (TCP)
2. User datagram protocol (UDP)

* TCP

→ Characteristics.

1. Reliable protocol : Guarantees the delivery of data packets to its correct destination. After receiving the packet, receiver sends an acknowledgement to the sender. TCP employs retransmission to compensate for packet loss.

2. Congestion & flow control : TCP handles congestion & flow control by controlling the window size. TCP reacts to congestion by reducing the sender window size.

3. Connection oriented protocol : TCP establishes an end to end connection between the source & destination. The connection is established before exchanging the data. The connection is maintained until the application programs at each end finishes exchanging the data.

4. In-order delivery : Sequence numbers are used to coordinate which data has been transmitted & received.

5. Full duplex : TCP connections are full duplex.

6. Collaboration with IP : A TCP connection is uniquely identified by combination of port numbers & IP addresses of sender & receiver. Port numbers are contained in the TCP header & 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.

7. Acknowledgement type : TCP can use both selective ↑ cumulative acknowledgements. TCP uses a combination of selective repeat & Go back N protocols. In TCP, $W_S = W_R$. 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 acknowledgements.

✓ To sum up, TCP is a combination of 75% SR & 25% GBN protocol.

✓ 8. 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 TCP header to the data chunk.

9. Error checking & recovery mechanism :

Using checksum, acknowledgement, retransmission.

→ TCP segment header.

Source Port Address 16 bits							Destination Port Address 16 bits	4B
Sequence Number 32 bits								4B
Acknowledgement Number 32 bits								4B
Header Length 4 bits	Reserved 6 bits	URG	ACK	PSH	RST	SYN	FIN	Window size (Advertisement window) 16 bits — 4B
Checksum 16 bits							Urgent pointer 16 bits	4B
Options & padding (0 - 40 bytes)								
Data (optional)								

min max
20 - 60 byte header ; followed by data from the application program.

(Unit of transmission in TCP - Segment).

1. Source port : Identifies the port of the sending application.

2. Destination port : Identifies the port of the receiving application.

TCP connection is uniquely identified using combination of port numbers & IP addresses.

3. Sequence number : Contains the sequence no. of first data byte. *

assigns unique seq. no. to each byte of data in TCP segment.

4. Acknowledgement no. : Sequence no. of the data byte that receiver expe to receive next from the sender. It is always sequence numbere of the last received data byte incremented by 1.

5. Header length : Contains the length of TCP header. It helps in knowing from where the actual data begins.

Min-Max HLen.

Length of TCP header always lies in the range [20B, 60B].

Initial 5 rows of the TCP header are alwa used. So, min length of TCP header 5×4 = 20B

Size of options field can go up to 40 B. So, max length (20+40) = 60 B.

* In TCP, we send chunk of data bytes. So, only the seq. no 1st data byte is put in seq. no. field

Scaling factor : As HLen is a 4 bit field, range is [0,15]. But, range of HLen is [20,60]. So, to represent the HLen, we use a scaling factor of 4.

✓ Header length = Header length field value × 4 Bytes.

eg. HLen field 0101 ≡ $(5)_{10}$

Header length = 5 × 4 = 20. bytes.

. Range of HLen field value [5, 15].

6. Reserved bits. : Not used.

7. URG bit : Used to treat certain data on an urgent bassis. When 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 has to be prioritized. Receiver forwards urgent data to the receiving application on a separate channel.

8. ACK bit : Indicates whether acknowledgement no. field is valid or not. When set, ACK no. is valid. For all TCP segments except request segment, ACK bit is set to 1.

✓ Except request segment, ACK bit is set to 1.

✓ Request segment is sent for connection establishment during Three-way handshake.

9. PSH bit : Used to push the entire buffer immediately to the receiving applicatio

When set, all the segments in the buffer are imm-ediately pushed to the receiving application. No wait is done for filling the entire buffer. This makes the entire buffer to free up immediately.

✓ Unlike URG bit, PSH bit does not prioritize the data. Same order is maintained while pushing, as of the arrival of segments.

It's not a good practice to set PSH to 1, as it disrupts the wooking of receiver's CPU & forces it to take an action immediately.

10. RST bit : Used to reset the TCP connection. When set, it indicates the receiver to terminate the connection immediately. It causes both the sides to release the connection & all its resources abnormally. It may result in the loss of data that is in transit. This is used only when, there are unrecoverable errors. There is no chance of terminating the TCP connection normally then

11. SYN bit : Used to synchronise the sequence numbers. When set, it indicates the ✓ receiver that the sequence no. contained in the TCP header is the initial sequence number. ✓ Request segment sent for connection establishment during Three way handshake contains SYN bit set to 1.

12. FIN bit : Indicates the end of data transmission to finish a TCP connection.

13. Window size : Contains the size of the receiving window of the receiver or sender. It advertises how much data (in bytes) the sender can send without acknowledgement (WR). Thus, window size is used for flow control.

No. of bytes that a receiver is willing to accept. rwnd is determined by the receiver. The sender must obey the dictation of the receiver in this case.

The window size changes dynamically during data transmission. It usually increases during TCP transmission up to a point where congestion is detected. After congestion is detected, the window size is reduced to avoid having dropped packets.

congestion control

14. Checksum : Verifies the integrity of data in the TCP payload. Sender adds CRC checksum to the checksum field before sending the data.

15. Urgent pointer : 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. This field is considered valid & evaluated only if URG is set.

urgent bytes = urgent pointer + 1.

✓ End of urgent byte =

Sequence no. of the first byte in the segment + Urgent pointer.

16. Optional fields. :

Used for following purposes -

✓✓ a) Time stamp - When wrap around time is less than life time of a segment, multiple segments having the same sequence number may appear at the receiver side. This makes it difficult for the receiver to identify the correct segment. If time stamp is used, it marks the age of TCP segments. Based on the time stamp, receiver can identify the correct segment.

✓ b) Window size extension - May be used to represent a window size greater than 16 bits.

c) Parameter negotiation - For example, during connection establishment, both sender & receiver have to specify their max. segment size. To specify maximum segment size, there is no special field. So, they specify their MSS using this field & negotiates. (MSS)

d) Padding - Addition of dummy data to fill up unused space in the transmission unit & make it ~~confirme~~ conform to the standard size is padding.

eg. HLen not multiple of 4. Extra zeros are padded in the options field. By doing so, HLen becomes multiple of 4.

HLen = 30 B. 2B dummy data

⇓

HLen 32 B.

HLen field $\frac{32}{4} = 8$

- In worst case, 3 bytes of dummy data might have to be padded to make the HLen a multiple of 4.

→ Well known ports used by TCP :

Port	Protocol	Description.
7	Echo	Echoes a received datagram back to the sender
9	Discard	Discards any datagram that's received
11	Users	Active users.
13	Daytime	Returns the date & the time
17	Quote	Returns a quote of the day
19	Chargen	Returns a string of characters.
20	FTP, Data	File transfer protocol (data conn^n)
21	FTP, control	" (control conn^n)
23	TELNET	Tenninal Network
25	SMTP	Simple Mail Transfer Protocol
53	DNS	Domain name server
67	BOOTP	Bootstrap protocol
79	Finger	Finger
80	HTTP	Hypertext Transfer Protocol
111	RPC	Remote procedure call

MSS - Window size. (buffer capacity of sender/receiver)

↓

S/R tells their MSS so when datagram comes to them there is no need of segmentation. MSS - options field

---

Assume, sender's window size = W_S Bytes

MSS of sender = 1460 B

MSS of receiver = ~~1460~~ 500 B

} We take the min of them to avoid segmentation

⇓

MSS = 500 B

Receiver's window size = W_R Bytes

∴ Receiver can send maximum of $\frac{W_S}{500}$ packets/segments

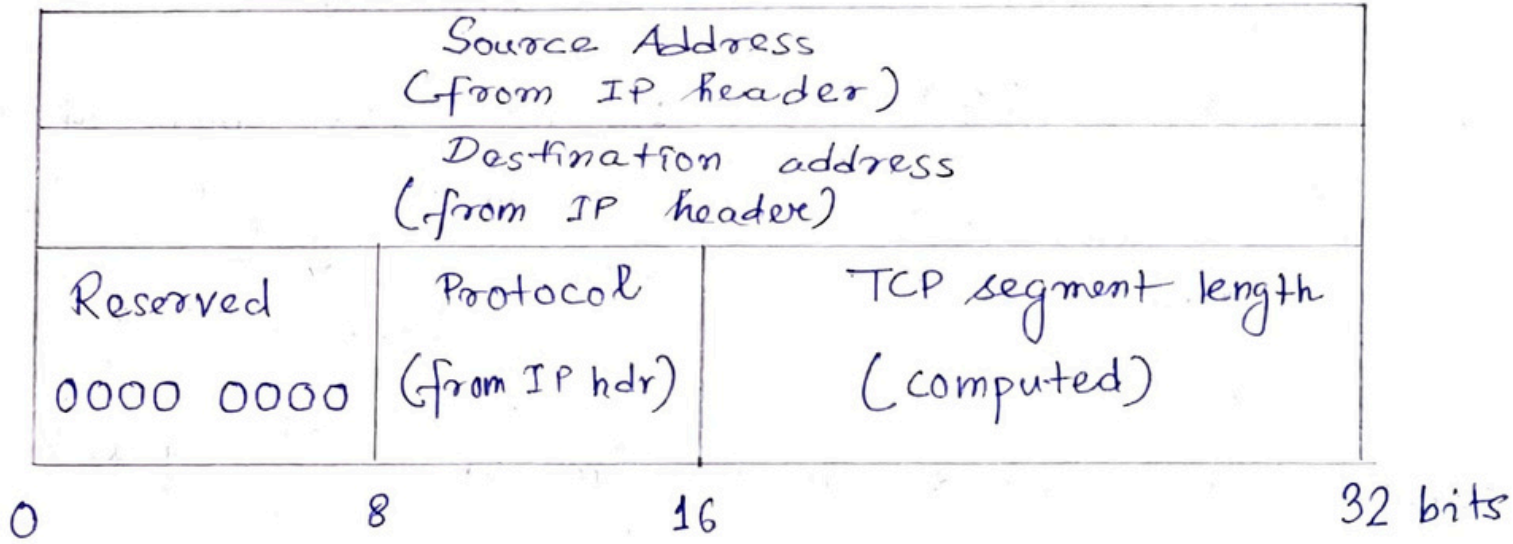
Sender can send maximum of $\frac{W_R}{500}$ segments

→ Receiver's sending window size

→ Sender's sending window size

Checksum (CRC) : 16b, TCP checksum is computed on TCP header, TCP data and pseudoheader.

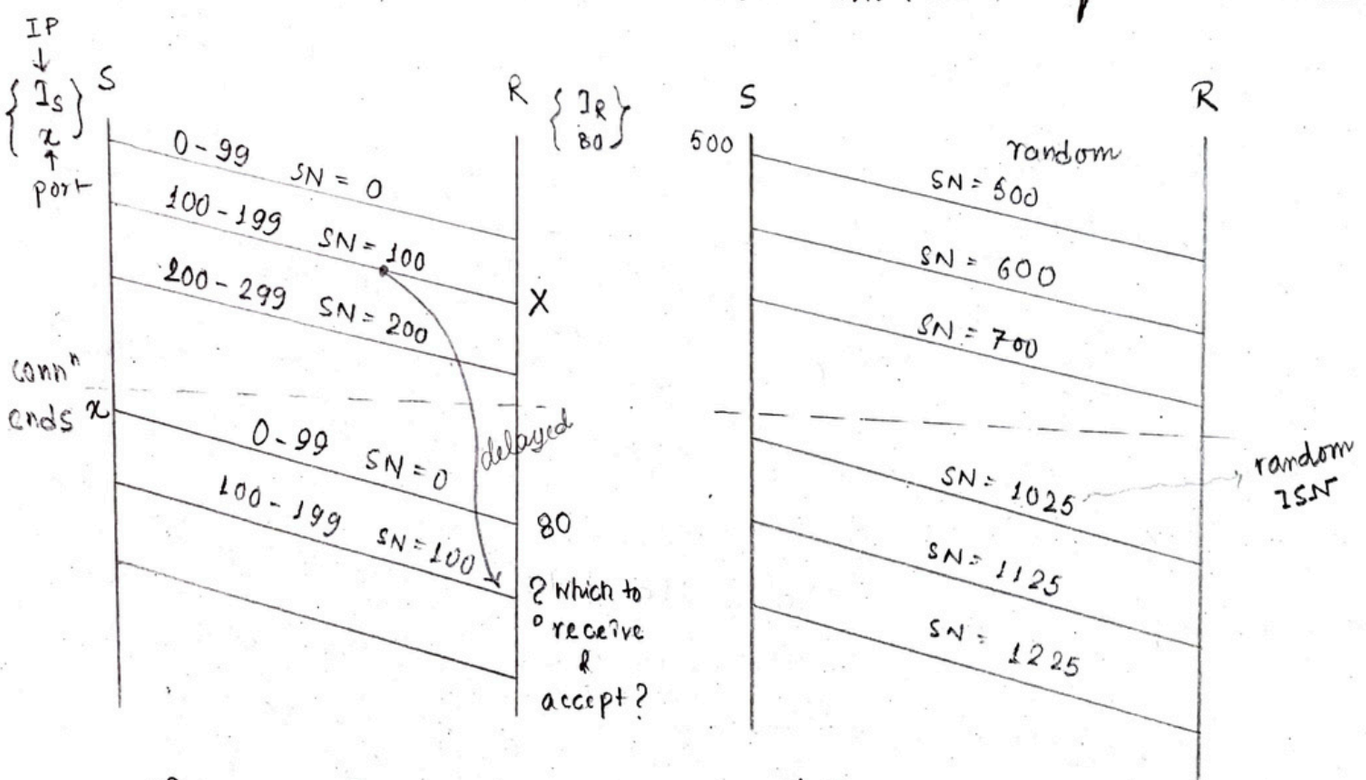
Pseudoheader — (used for double checking)



- **Sequence number, Acknowledgement Number. :**

To ensure connectivity, each byte to be transmitted is numbered. During connection establishment each party uses a random number generator to create initial sequence number (ISN) which is usually different in each direction. TCP sequence number field is 32 bits. So, it has 4 Giga sequence numbers and as TCP is a byte stream protocol, we will be able to send only 4 GB of data with unique sequence number.

Reason to use random initial sequence number-



- Purpose of sequence numbers: i) helps to identify each data byte uniquely. ii) helps in the segmentation of data into TCP segments & reassemble them later. iii) Keeping track of how much data has been transferred & received. iv) put the data back into the correct order if it is received in the wrong order. v) Request data when it has been lost in transit.

- Maximum number of possible sequence numbers = 2^{32} = 4G
  Concept of wrap around allows to send unlimited data using TCP.

- Wrap around.

  After all the 2^{32} sequence numbers are used up & more data is to be sent, the sequence numbers can be wrapped & used again from starting.

  In general, if ISN chosen is X, sequence numbers are used from X to $2^{32}-1$ and then from 0 to X-1. Then, seq. no.s are wrapped.

  - Wrap around time: Time taken to use up all the 2^{32} sequence numbers. It depends on the bandwidth of the N/W i.e. the rate at which the bytes go out. More the bandwidth, lesser the WAT & vice versa.

$$WAT \propto \frac{1}{BW}$$

$$WAT = \frac{\text{No. of sequence numbers available.}}{\text{Bandwidth in Bytes/sec.}}$$

  - Life time of TCP segment: Life time of a TCP segment is 180 seconds (3 m). After sending a TCP segment, it might reach the receiver taking 3 min in worst case.

WAT must be greater than the life time to avoid same sequence numbers for bytes.

When WAT ≥ LT, by the time the seq no.s wrap around, there's no probability of existing any segment having the same sequence no.

- To reduce the WAT to the LT, there must exist as many seq. no.s as there are no. of data bytes sent in time equal to the LT of segment.

After previous segment reaches receiver, wrap around to avoid confusion of same SNs

~ No. of seq. no.s needed if WAT is t and BW B bytes/sec = Bt.

~ # bits reqd. in the seq. no. field so that wrap around time becomes equal to lifetime of TCP segment =

$$\log_2 (\text{LT of TCP segment} \times \text{BW}).$$

Extra bits needed are accomodated in the Options field of the TCP header.

Q. BW = 1 MB/s. WAT = ?

→ $$\text{WAT} = \frac{\#\text{ seq. no. available.}}{\text{BW}} = \frac{2^{32}}{10^6 \text{ Bps}} = 1.19 \text{ hr}$$

Q. BW = 1 GBps, how many extra bits have to be appended in the options field so that WAT = LT ?

→ $$\log_2 (\underbrace{180 \times 2^{30}}_{\text{bytes sent}})$$

data ▶ power of 2
BW ▶ power of 10

$\simeq 38$

ans $(38 - 32) = 6$ bits.

BW 1 GBps
bytes transferred in 1s = 1 GB
" " in 180 s = 180 GB
$= 180 \times 2^{30}$ B
∴ # of seq. no reqd = 180×2^{30}

WAT = LT

$\frac{\#\text{seq}}{\text{BW}} = \text{LT}$

$\#\text{seq} = \text{BW} \times \text{LT}$

Q. In a N/W that has a maximum TPDU (transport layer protocol data unit / segment) size of 128 bytes, a max. TPDU lifetime of 30 sec & 8 bit seq. no., What's the max. data rate per connection?

→ Max. amount of data that can be sent in 30 sec = 2^8 = 256 bytes.

Max data rate per connection = $\frac{256 \text{ bytes}}{30 \text{ s}}$ = 8.5 byte/s.

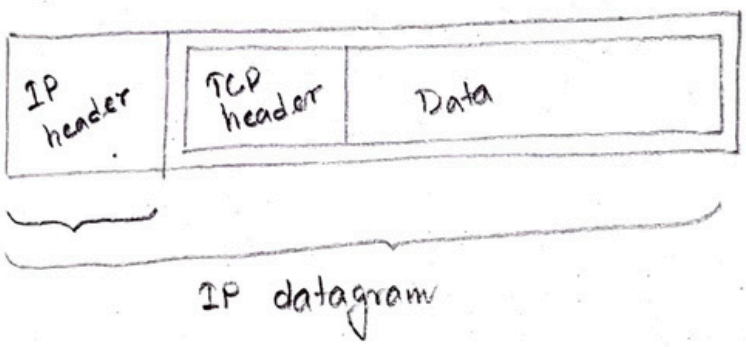
Q. Advertised window is 1 MB long. If a seq. no. is selected at random from the entire seq. no. space, what is the probability that the seq. no. falls inside the advertised window?

→ $\frac{2^{20}}{2^{32}} = \frac{1}{2^{12}}$.

Window size = 2^{20} Bytes

2^{20} seq. no.

→ Header length and calculation of Ack. no.



IP datagram total length be 1000B

IP header 5 B.

TCP header 5B.

Seq. no. of 1st byte in TCP segment 100.

What is the ack no. sent from the receiver?

Header size = 5 x 4 = 20 B [as min header size is 20 B].

Data in TCP segment = (1000 - 20 - 20) = 960 B.

Last byte seq. no. = 100 + 960 - 1 = 1059

Ack no. = 1060 (Ans)

→ TCP connection establishment

(3 way handshake.)

TCP establishes an end to end connection between the sender & receiver. This connection is established between them before exchanging the data.

3 way handshake is a process used for establishing a TCP connection. Client wants to establish a connection with the server. Before 3 way handshake, both client & server are in closed state.

Steps:

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 —

i) Initial sequence number (RISN)
ii) SYN bit set to 1.
iii) Maximum segment size
iv) Receiving window size of sender

(i) Client sends the ISN to the server. Contained in the sequence no. field. Randomly chosen 32 bit value.

(ii) Client sets SYN bit to 1 which indicates the server that the segment contains the ISN used by the client. It has been sent for synchronising the sequence numbers.

→ SYN flooding attack: A malicious attacker sends a large number of SYN segments to a server, pretending that each of them is coming from a different client by faking the source IP addresses in the datagrams. The server, assuming that the clients are issuing an active open, allocates the necessary resources. This SYN flooding attack belongs to a type of security attack known as a denial-of-service attack, in which an attacker monopolizes a system with so many service requests that the system collapses & denies service to every request.

(iii) MSS dictates the size of the largest data chunk that client can send & receive from the server. Contained in the options field.

(iv) Limit of unacknowledged data that the client can receive.

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 following information -

(i) ISN ~ Server sends the ISN to the client.

(ii) SYN bit set to 1 ~ Indicates the client the segment contains ISN

(iii) MSS ~ Size of largest chunk that server can send & receive from the client.

(iv) Receiving window size ~ Limit of unack. data that the server & buffer can receive. Contained in the window size field.

(v) Acknowledgement no. ~ Server sends the ISN incremented by 1 as an ack. no.

(vi) ACK bit set to 1 ~ Server sets ACK bit to 1. Indicates the client that the ack. no. field in the current segment is valid.

3. ACK - After receiving the reply segment, client acknowledges the response of server. It acknowledges the server by sending a pure acknowledgement.

This is just an ACK segment. Sequence no. for this segment is that of client's ISN incremented by 1. But, this same sequence will be used for data transfer. Hence, we can say that this ack segment does not consume any sequence number.

- A SYN segment cannot carry data but it consumes one sequence number.

- A SYN + ACK segment cannot carry data, but does consume one sequence no.

- TCP connection establishment phase consumes 1 seq no. of both the sides.

Request (SYN) segment consumes 1 seq. no. of the requester.

Reply segment (SYN + ACK) consumes 1 seq. no. of the respondent.

Pure acknowledgements (ACK) don't consume any seq. no.

- Pure acknowledgement for the reply segment is not necessary. This is because - if client sends the data packet immediately then it will be considered as an ack. It means that on the first two steps only, the full duplex connection is established.

- For all the segments except the request, ACK bit is always set to 1.

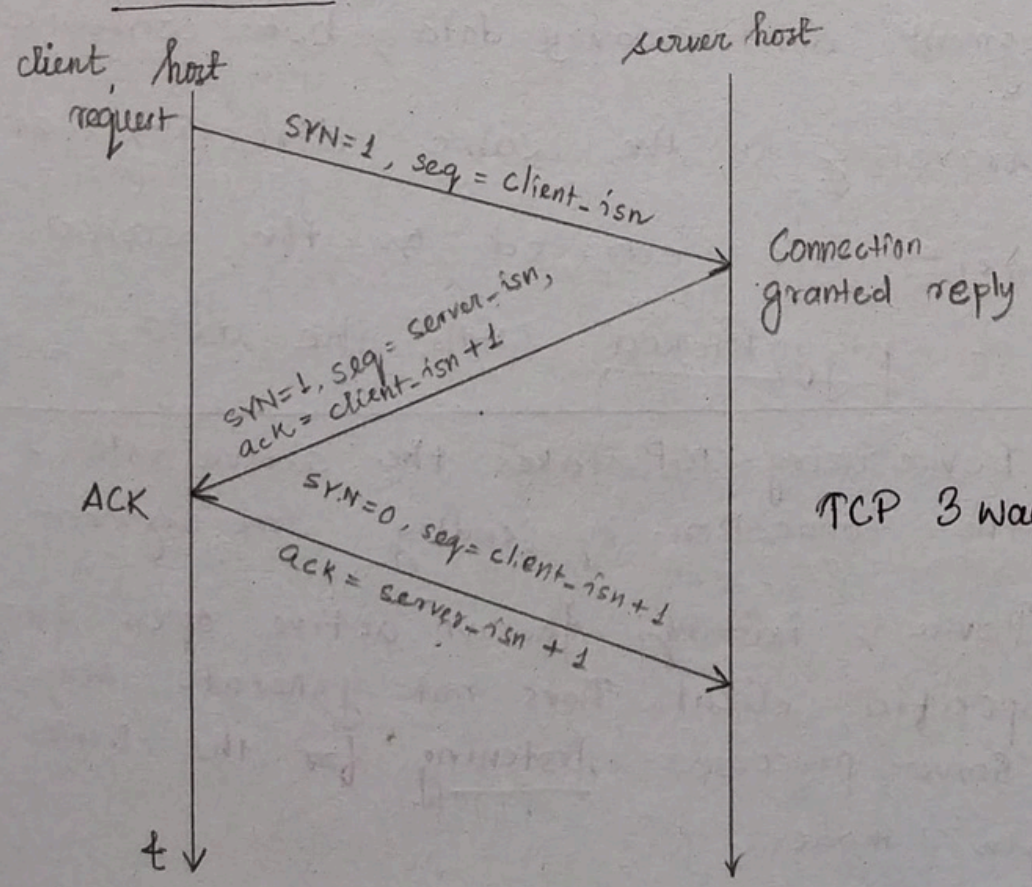
- Certain parameters (window size, MSS, timer values) are negotiated during conn^n establishment.

- In any TCP segment :

SYN	ACK	Segment remark
1	0	Request segment
1	1	Reply segment
0	1	Can be pure ACK or segment meant for data transfer.
0	0	Not possible. (as except SYN request segment ACK bit is always 1).

- There is no dedicated field for MSS in TCP header. This is because MSS has to be informed only once. So, if dedicated field would be present, then sending it each time would not be required. So, MSS is informed once using Options.

- An ack. by TCP sender guarantees data has been delivered to the application.



TCP 3 way handshake.

(namaste!
corona alert ⚠)

- SYN packet consumes 1 sequence no.

ACK=1 consumes 0 seq. no.

FIN=1 consumes 1 seq. no.

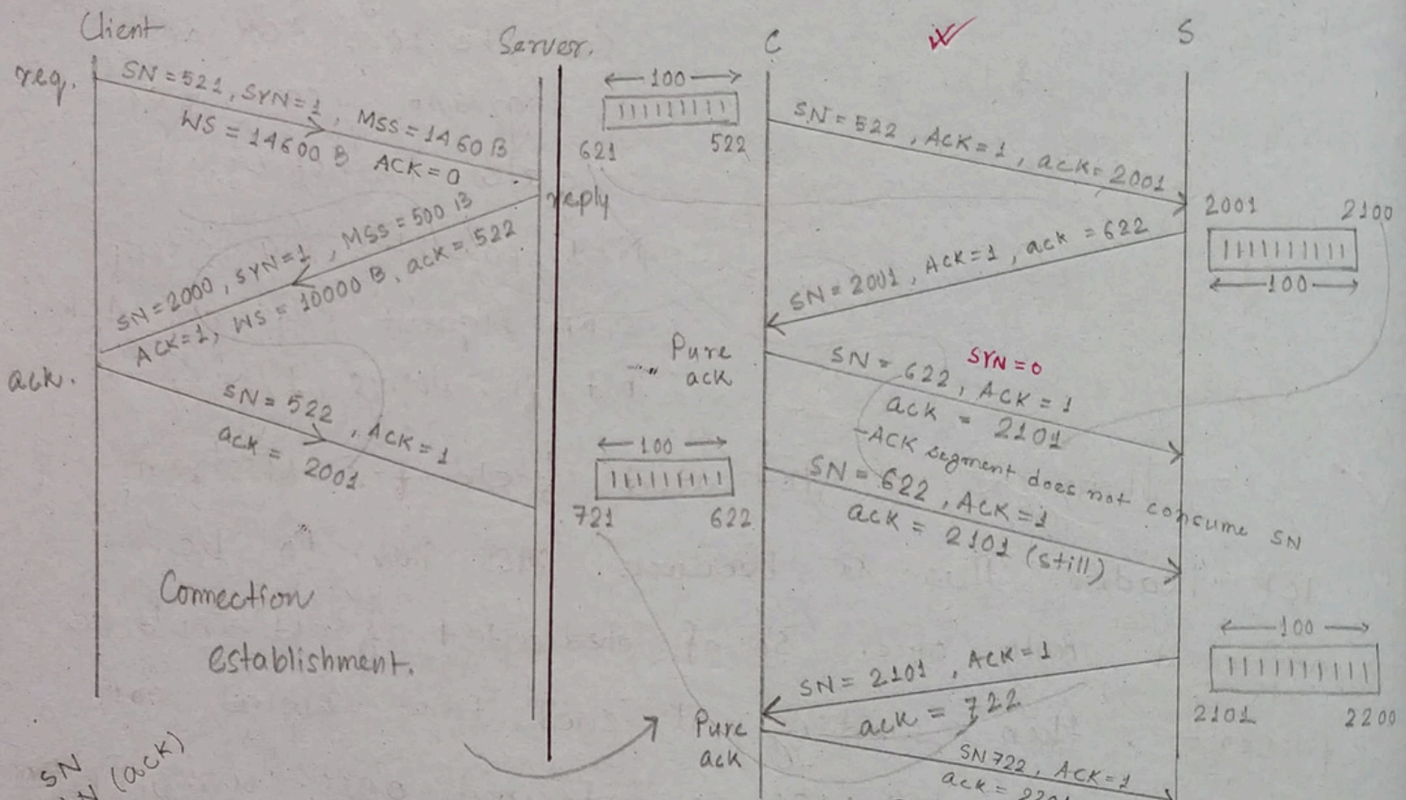
1 data byte consumes 1 seq. no.

Full duplex TCP

Pure ack - does not consume seq. no.

→ Data transfer after connection establishment.

SN - seq. no.
WS - window size



Data transfer.

SN
AN (ack)
SYN
ACK
WS
MSS
FIN

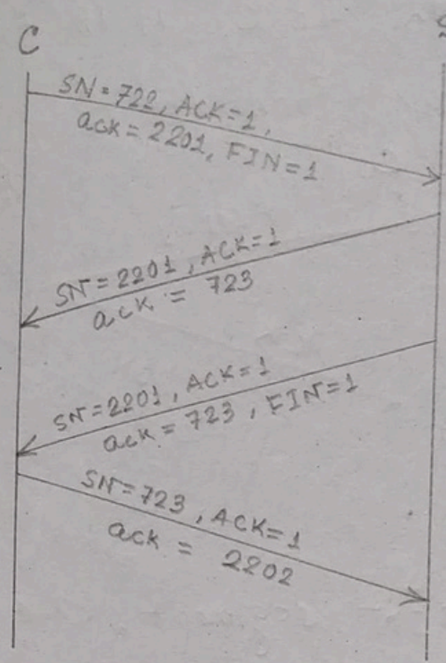
- An ACK segment, if carrying no data, consumes no SN. A SYN or SYN+ACK segment don't carry data, but consumes one SN.

✓ - Data traveling in the same direction as an acknowledgement are carried on the same segment. The ack. is piggybacked with the data.

Active open : Device using TCP takes the active role & initiates the connection by sending SYN segment.

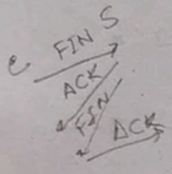
Passive open : Device is waiting for an active open from a specific client. Does not generate any TCP msg segment. Server processes listening for the clients are in passive open mode.

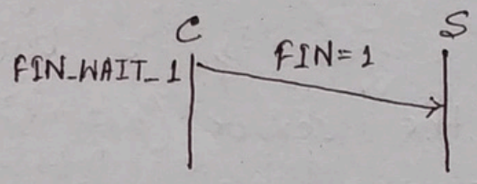
→ TCP connection termination.



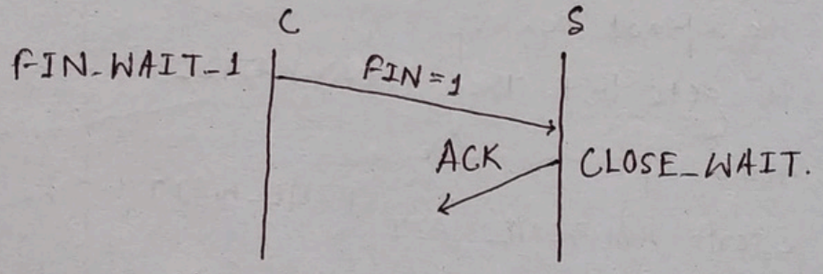
A TCP connection is terminated using FIN segment where FIN bit is set.

Step 1. For terminating a conn^n client sends a FIN segment to the server with FIN bit set. Client enters the FIN_WAIT_1 state as it waits for an ack. from the server.

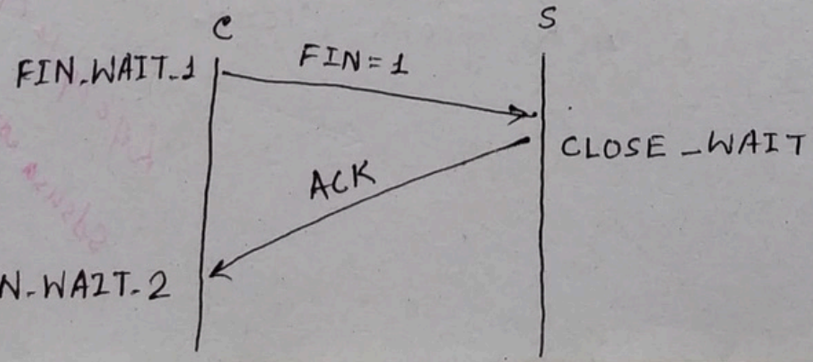




Step 2 After receiving the FIN segment, server frees up its buffers. Server sends an ack to the client. Server enters the CLOSE_WAIT state.

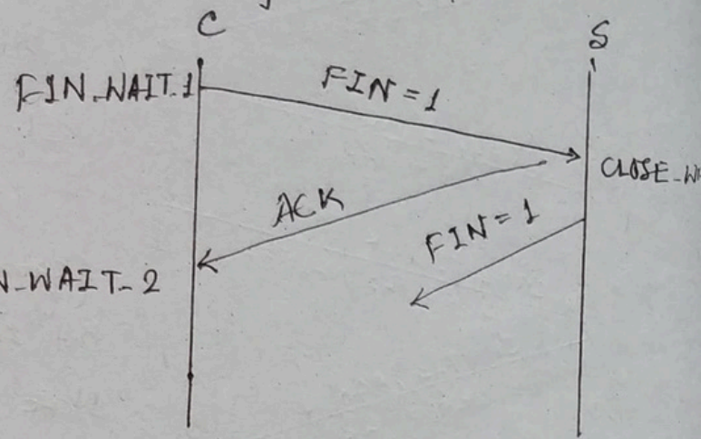


Step 3 After receiving the ack, client enters the FIN_WAIT_2 state. Now, the conn^n from C to S is terminated i.e. one way comm^n is closed. Client can't send any data to the server since server has released its buffers. Pure ack.s can still be sent from the C to S. The comm^n from S to C is still open. Server can send both data & ack.s (not needed as no data from C to S) to the client.



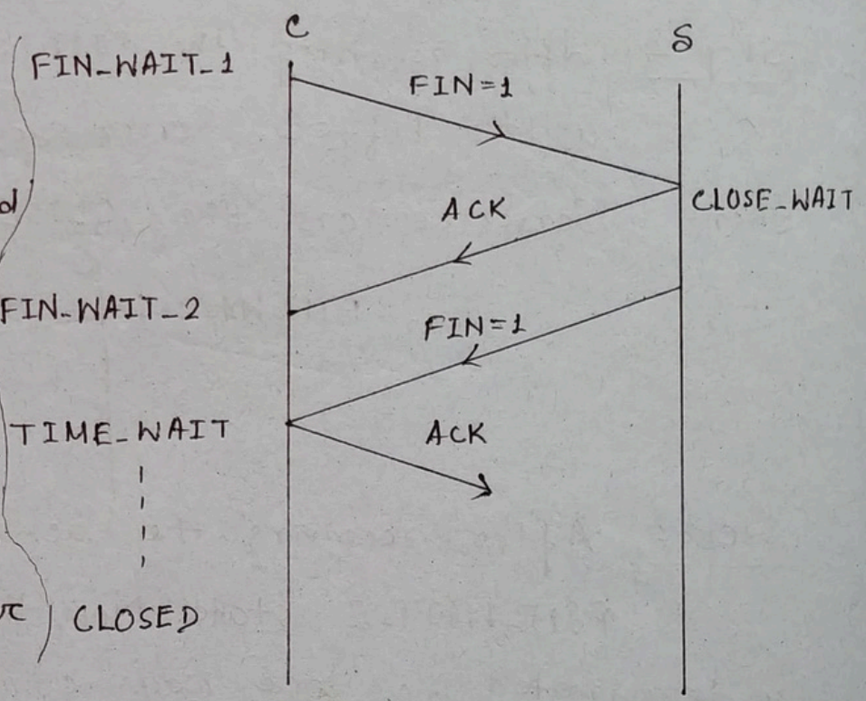
Step 4 Now, if S wants to close the conn^n with C, S sends a FIN segment to the C with FIN bit set. S waits for an ack from the C.

If server wanted, it could have sent the FIN segment along with the previous ack. that it sent to the client. (piggybacking)



Step 5 After receiving the FIN segment, client frees up its buffers. Client sends an ack. to the server (not mandatory). Client enters the TIME_WAIT state.

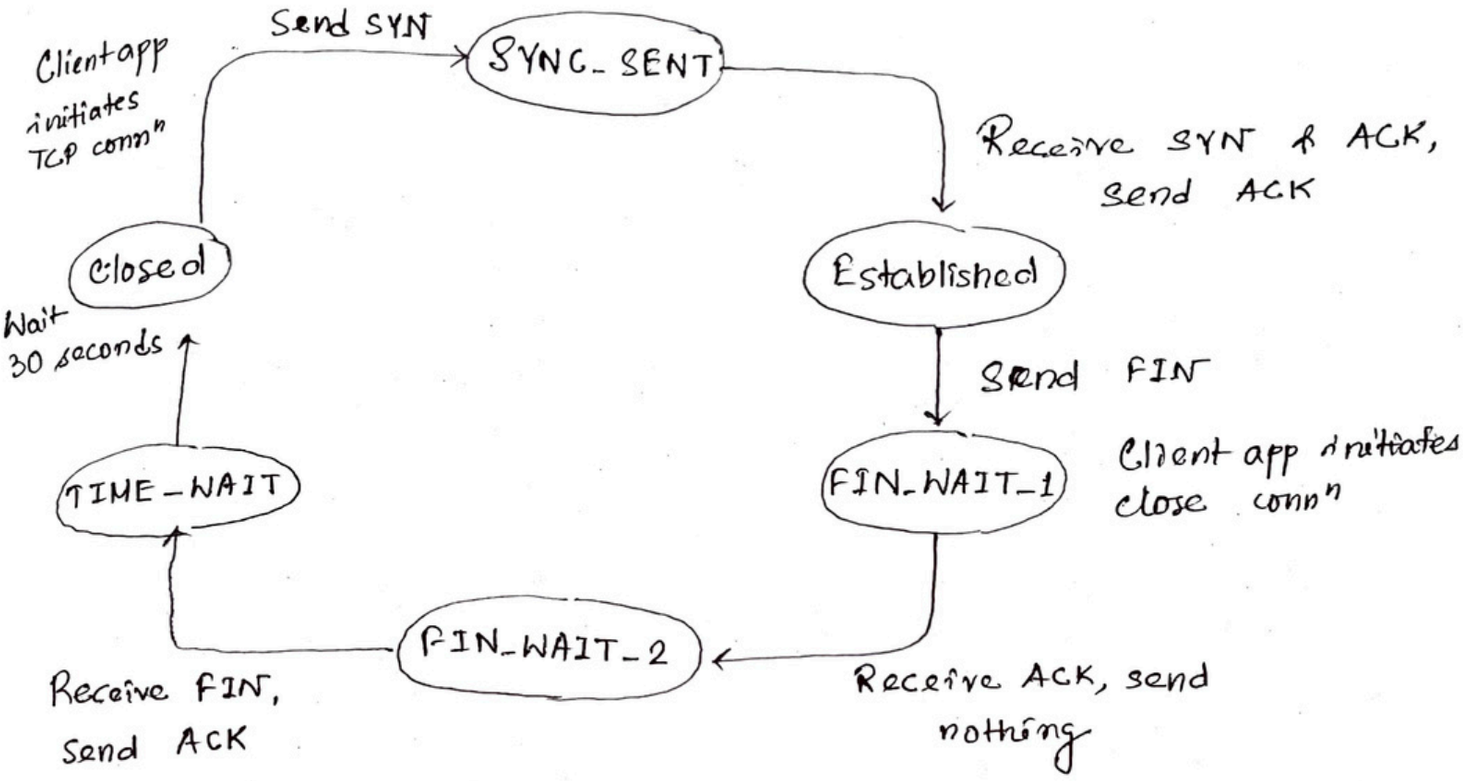
The TIME_WAIT state allows the C to resend the final ack if it gets lost. The time spent by the client in TIME_WAIT state depends on the implementation. After the wait, the conn^n gets formally closed.



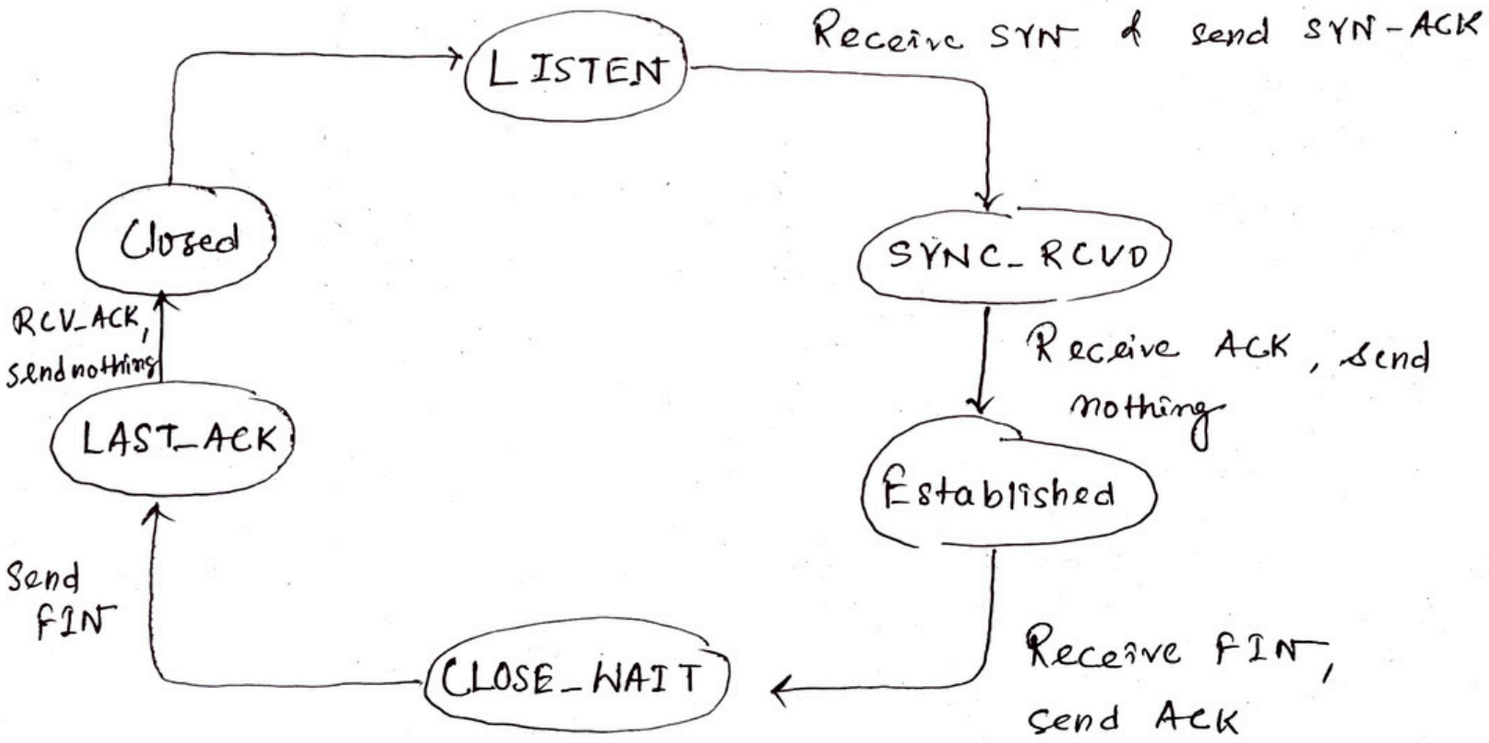
← 1. FIN from client
2. ACK from server
3. Client waiting
4. FIN from server
5. ACK from client.

Life cycle of TCP/IP
sdsusa.com/support/connections/

TCP states visited by Client -



TCP states visited by server -

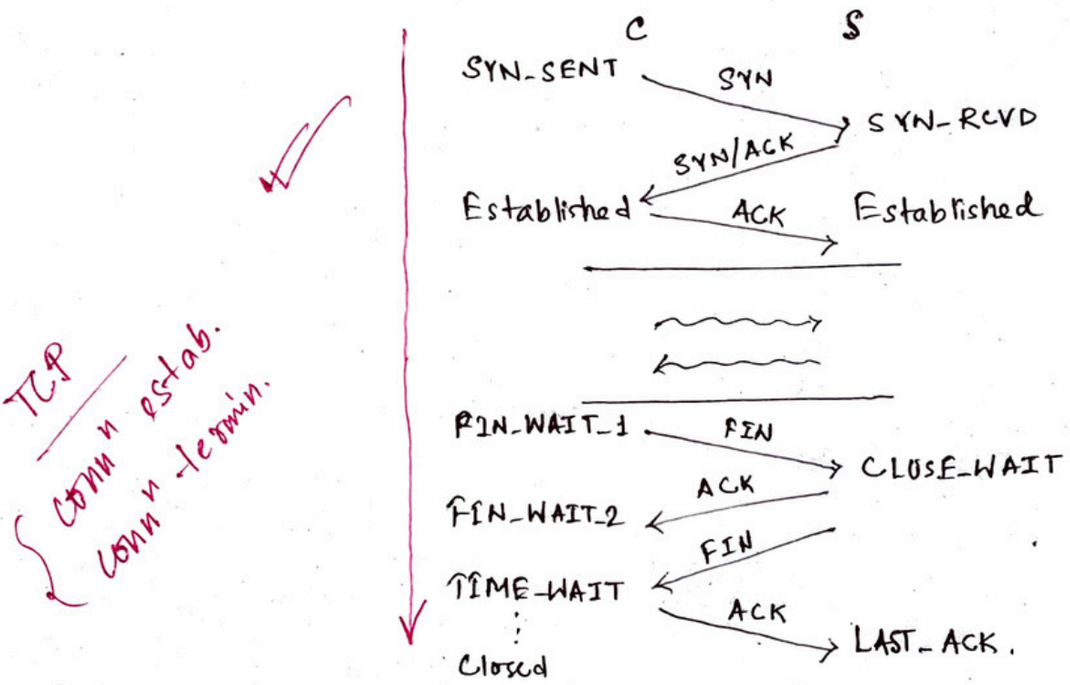


Tanenbaum (4e) 415p

Q. G'17 Consider a TCP client & a TCP server running on 2 different machines. After completing data transfer, the TCP client calls 'close' to terminate the conn^n & a FIN segment is sent to the TCP server. Server responds by sending an ACK which is received by client. As per the TCP connection state diagram (RFC 793), in which state does the client side TCP conn^n wait for the FIN from the server-side TCP?

a) LAST_ACK
b) TIME_WAIT
c) FIN_WAIT_1
d) FIN_WAIT_2.

DDOS - Distributed denial of service (attacks)



TCP
conn estab.
conn termin.

PDU SDU service data unit

Message AL
Segment TL
Datagram NL
Frame DLL
PDU PL
↳ protocol data unit

segment/datagram
packet
frame
bit (generally symbol)

wiki

TCP - UDP (contd)

→ PSH flag: Transport layer by default waits for some time for application layer to send enough data equal to maximum segment size so that the no. of packets transmitted on N/W is minimized which is not desirable by some application like interactive ones. Similarly transport layer at receiver end buffers packets & transmit to application layer if it meets certain criteria.

This problem is solved by using PSH. Transport layer sets PSH = 1 & immediately sends the segment to N/W layer as soon as it receives signal from application layer. Receiver transport layer, on seeing PSH = 1 immediately foowards the data to application layer. In general, it tells the receiver to process these packets as they are received instead of buffering them. The sending TCP must not wait for the window to be filled. It must create segment & send it immediately.

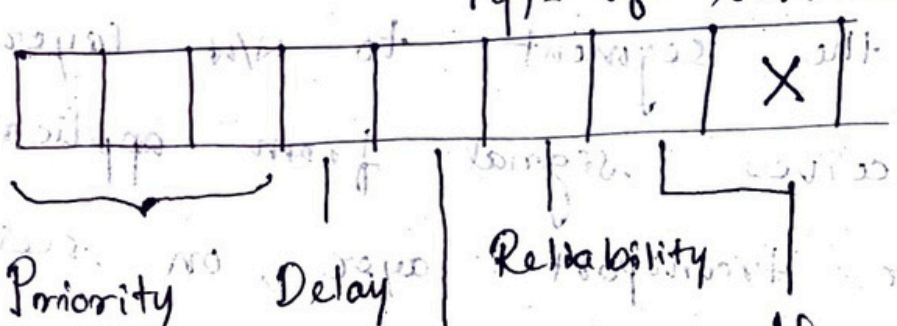
→ URG flag: Data inside a segment with URG = 1 is forwarded to application layer immediately even if there are more data to be given to application layer. It is used to notify the receiver to process the urgent packets before processing

For interactive applications (chat) we need to transfer data to the application layer more often to keep the app interactive. Then, the PSH flag is set to 1.

all other packets. The receiver will be notified when all known urgent data has been received.

How IPv4 sets urgent data?

In IPv4, types of service field, first 3 bits are to set priority. As routers don't have transport layer, to make the data urgent we need to set priority value as 7 (111) (TCP is transport layer protocol).



Data that is urgent -

(Seq. no.) to (SN + URG pointer value)

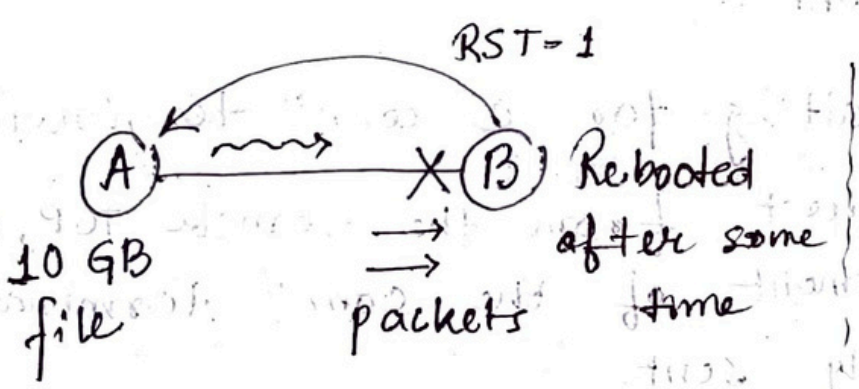
→ PSH

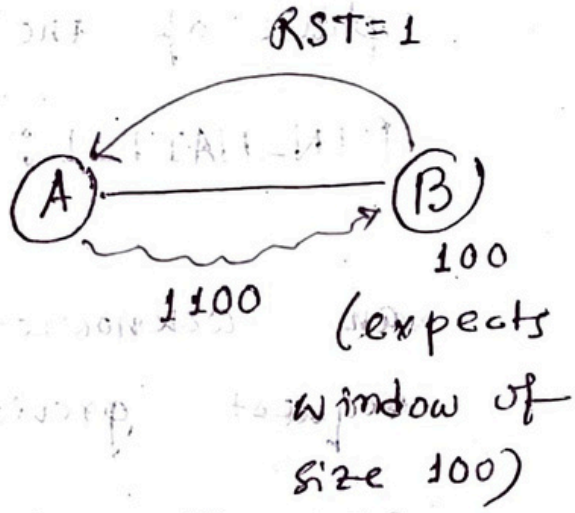
- All data in buffer to be pushed to NL (sender) or AL (receiver).
- Data delivered in order.
- No priority is set.

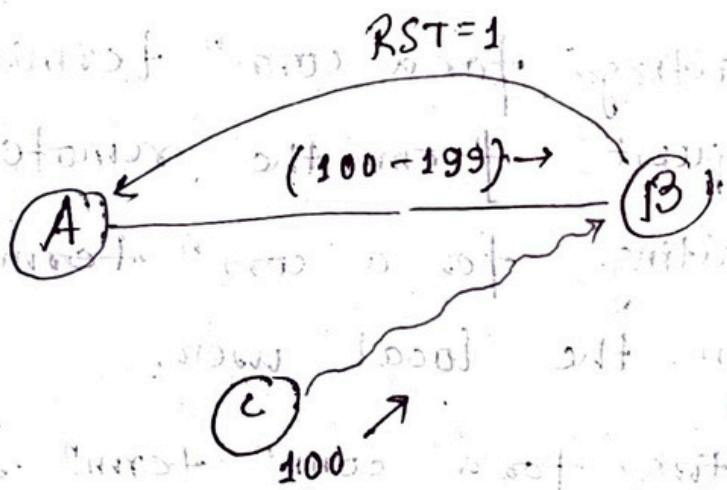
URG

- Only the urgent data to be given to AL immediately.
- Data delivered out of order.
- Priority set.

→ RST flag : Used to terminate the conn^n if the RST sender feels something is wrong with the TCP conn^n or that the conversation should not exist.







expecting SN = 100
now expects 200 but receives 100 from C
so RST = 1

C sending as A (pretends)

→ TCP state transition diagram.

Meaning of states:

LISTEN : Waiting for a connection request from (server) any remote TCP & port.

SYN_SENT : Waiting for a matching conn^n request after having sent a conn^n request.

SYN_RECEIVED : Waiting for a confirming conn^n request acknowledgement after having both received & sent a conn^n request.

Established : An open comm^n, data received can be delivered to the user. The normal state for the data transfer phase of the comm^n.

FIN_WAIT_1 : Waiting for a conn^n termination request from the remote TCP, or an acknowledgement of the comm^n termination request previously sent.

FIN_WAIT_2 : Waiting for a comm^n termination request from the remote TCP.

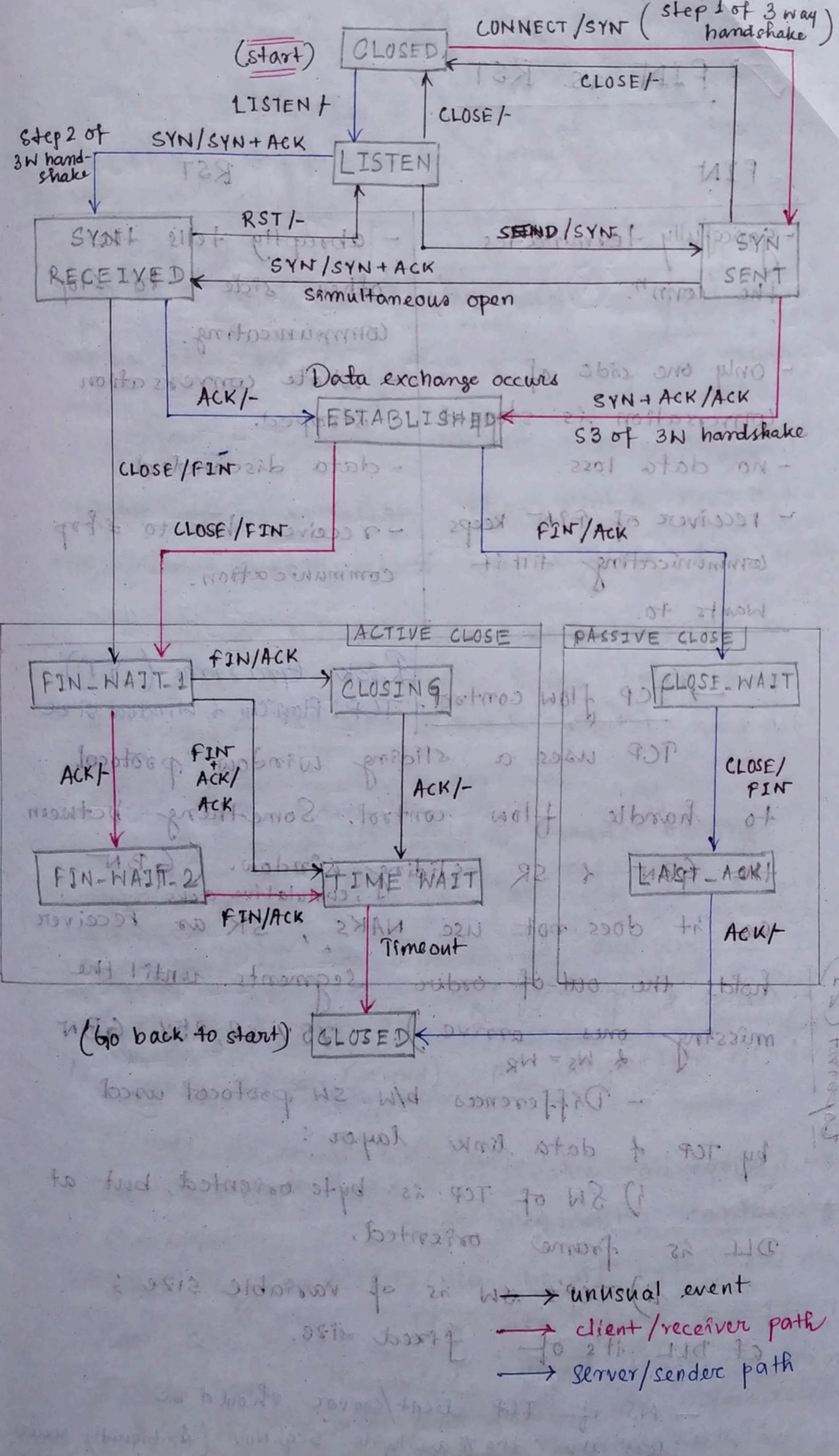
CLOSE_WAIT : Waiting for a comm^n term^n request from the local user.

CLOSING : Waiting for a comm^n term^n request acknowledgement from the remote TCP.

LAST_ACK : Waiting for an ack. of the comm^n term^n request previously sent to the remote TCP (that includes an ack of its comm^n term^n request).

TIME_WAIT : Waiting for enough time to pass to be sure the remote TCP received the ack. of its comm^n term^n request.

CLOSED : No connection state at all.


(start)
CLOSED
CONNECT/SYN (step 1 of 3 way handshake)
LISTEN/-
CLOSE/-
CLOSE/-
step 2 of 3W hand-shake
SYN/SYN+ACK
LISTEN
RST/-
SEND/SYN
SYN RECEIVED
SYN SENT
SYN/SYN+ACK
simultaneous open
Data exchange occurs
ACK/-
ESTABLISHED
SYN+ACK/ACK
S3 of 3W handshake
CLOSE/FIN
CLOSE/FIN
FIN/ACK
ACTIVE CLOSE
PASSIVE CLOSE
FIN_WAIT_1
FIN/ACK
CLOSING
CLOSE_WAIT
ACK/-
FIN+ACK/ACK
ACK/-
CLOSE/FIN
FIN_WAIT_2
TIME WAIT
LAST_ACK
FIN/ACK
Timeout
ACK/-
(Go back to start)
CLOSED
unusual event
client/receiver path
server/sender path

→ FIN vs. RST

FIN	RST
- gracefully terminates the comm^n.	- abruptly tells the other side to stop communicating.
- only one side of conversation is stopped.	- whole conversation stopped.
- no data loss.	- data discarded.
- receiver of FIN keeps communicating till it wants to.	- receiver has to stop communication.

→ TCP flow control.

Rick. Graziani (YT)
TCP: Flow Con. & Window size

TCP uses a sliding window protocol to handle flow control. Something between the GBN & SR sliding window. (GBN as it does not use NAKs, SR as receiver holds the out-of-order segments until the missing ones arrive.). 75% SR, 25% GBN
, cumulative acks
& $W_S = W_R$

- Differences b/w SW protocol used by TCP & data link layer:

i) SW of TCP is byte oriented, but at DLL is frame oriented.

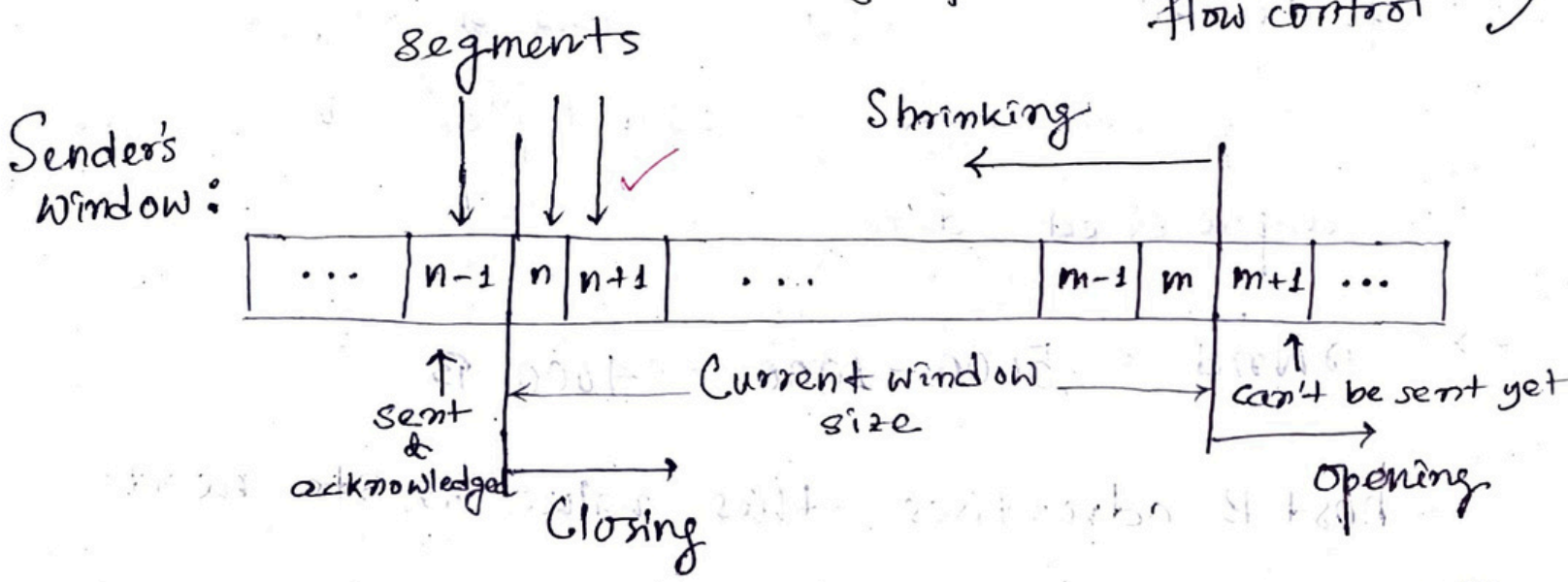
ii) TCP's SW is of variable size; at DLL it's of fixed size.

- HS of TCP client/server should be less than the # available Seq. No. (Ambiguity occurs otherwise)

topper (21)

The window is opened, closed or shrunk.

(Tutpoint YT : Sliding window flow control)



Opening a window means moving the right wall to the right. Allows more new bytes in the buffer that are eligible for sending

Closing the window means moving the left wall to the right. Some bytes have been acknowledged & the sender need not worry about them anymore.

Shrinking the window means moving the right wall to the left.

- Size of the window at one end is determined by the lesser of 2 values - receiver window (rwnd) or congestion window (cwnd).

(The rwnd is the value advertised by the opposite end in a segment containing acknowledgement. It's the no. of bytes the other end can accept before its buffer overflows & data are discarded. Cwnd limits the amount of data the TCP can send into the N/W before receiving an Ack.)

determined by N/W to avoid congestion.

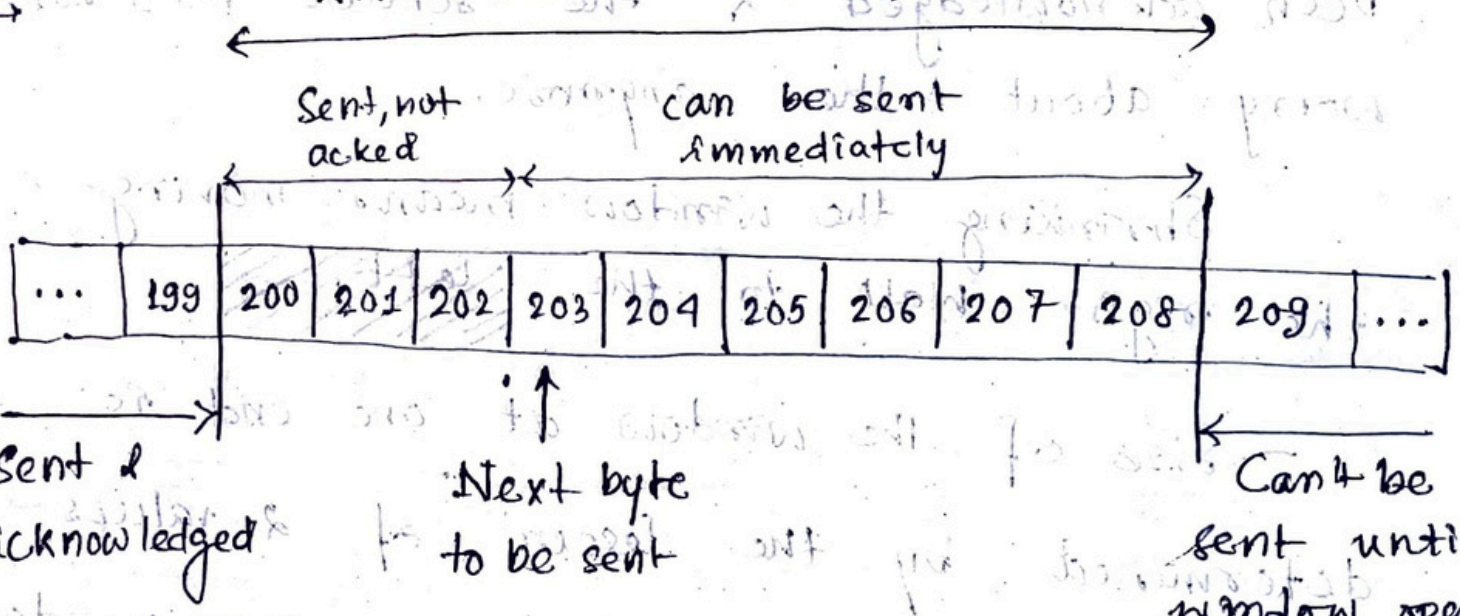
Q What is the value of rwnd for host A if the receiver, host B, has a buffer size of 5000 Bytes & 1000 Bytes of received & unprocessed data?

→ rwnd = 5000 − 1000 = 4000 B

host B advertises this value in its next segment to A.

Q. Sender has sent bytes upto 202. We assume that cwnd is 20. Receiver has sent an ack no. of 200 with an rwnd of 9 bytes. How many bytes can be sent now?

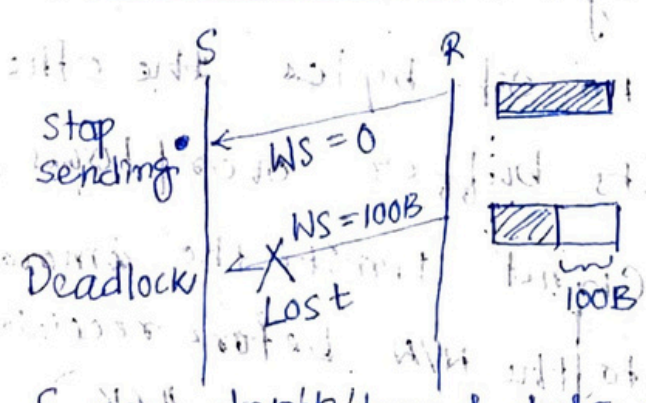
→ window size = min (20, 9) = 9



Bytes 203 to 208 can be sent without worrying about acknowledgement.

---

* Deadlock in TCP:



⇒ Solution: S sets persistent timer & when it goes off, sends a probe (1B size) to R. If R has WS > 0, it advertises in next packet.

S now doesn't send data as it does not know R's WS (as the packet was lost

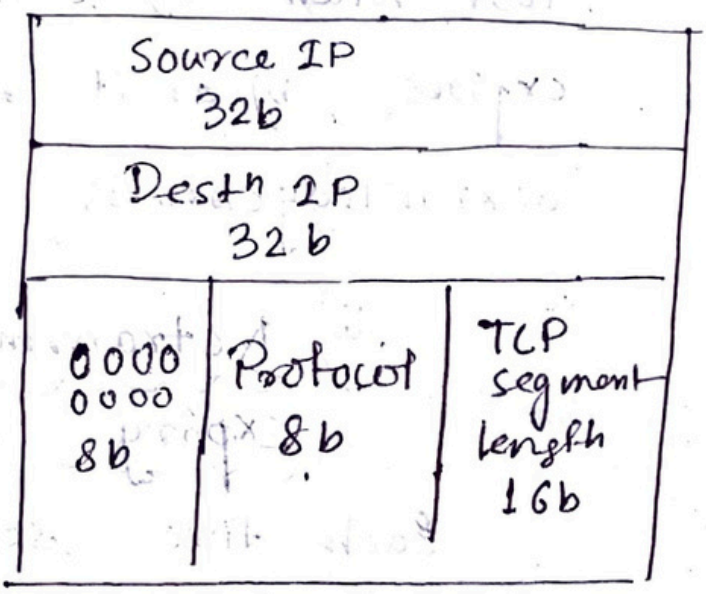
→ TCP Error control.

i) Checksum. : In TCP, the checksum is calculated both on the data & header.

Checksum is calculated on TCP header, TCP data, IP header. (We calculate checksum only on the fields in pseudo IP header).

Pseudo IP header

TCP header
+
TCP data
+
Pseudo IP header



ii) Acknowledgement. : TCP uses ack. to confirm the receipt of data segments. Control segments that carry no data but consume a sequence number are also acknowledged. ACK segments are never acknowledged.

iii) TCP retransmission. : After establishing the connⁿ, 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 ack with same ACK to the sender. As a result, sender

retransmits the same segment to the receiver. This is called TCP retransmission.

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

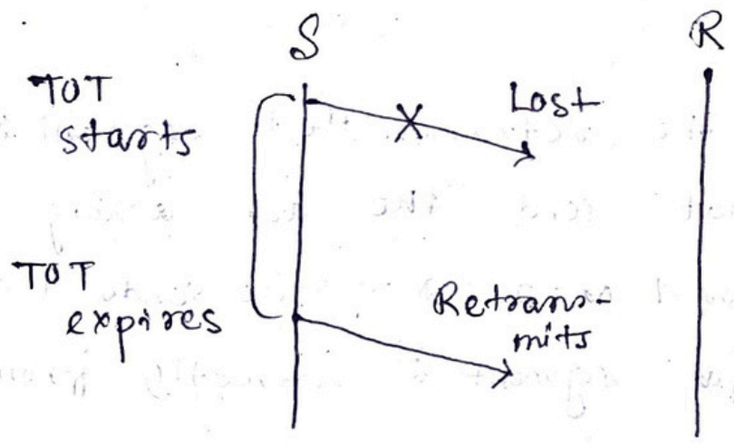
Sender discovers that the segment is lost when a) either time out timer expires, b) or it receives 3 duplicate acknowledgements.

a) Retransmission after TO timer expiry.

Each time sender transmits a TCP segment to the receiver, it starts a TOT. Now, 2 cases may arise —

(-1) Sender receives an ack for the sent segment before the timer goes off: Sender stops the timer.

(-2) Sender does not receive any ack. for the sent segment & timer goes off. In this case, sender assumes that the sent segment is lost. Sender retransmits the same segment to the receiver & resets the timer.

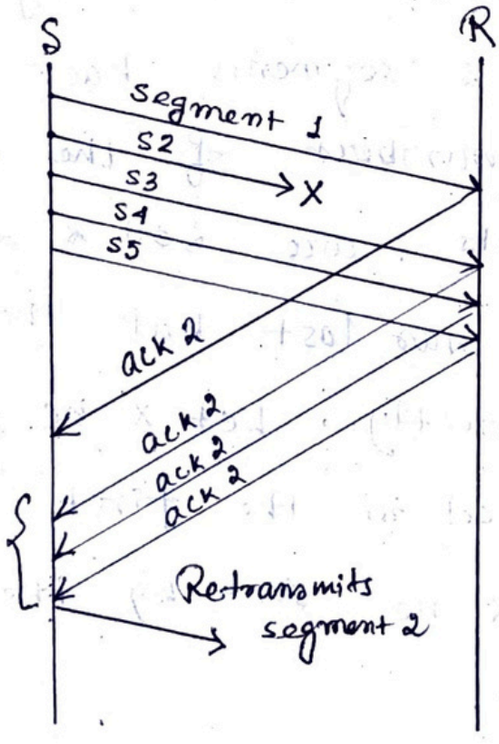


→ If 3 or more duplicate acks are received in a row it's a strong indication that a segment has been lost. (See stackexchange - why only 3 - duplicate acks...)

b) Retransmission after receiving 3 duplicate acknowledgements.

Sender receives 3 duplicate ack.s for a TCP segment sent by it. Then, sender assumes that the corresponding segment is lost. So, Sender retransmits the ~~the~~ same segment without waiting for its TOT to expire. This is called <u>early or fast retransmission.</u>

eg. Sender sends 5 TCP segments to the receiver. The second TCP segment gets lost before reaching the server.



In TCP we send ack when all segments with SNT ack no. are accepted.

ack 300 ↓ segment 1. 0-99 ✓ all 3 accepted
2 100-199 ✓
3 200-299 ✓

After receiving the retransmitted segment 2, receiver does not send the ack. asking for segment-3 or 4 or 5. Receiver sends the ack. asking for segment 6 directly from the sender. This is because previous segments have been already received & acknowledgements for them have been already sent (although wasted in asking for segment 2).

✓ - TOT expires before receiving the ack for a TCP segment → stronger possibility of congestion in the N/W.

✓ Sender receives 3 duplicate ack.s for same segment → weaker possibility of congestion in the N/W.

Q Consider a TCP connection in a state where there are no outstanding ACKs. The sender sends 2 segments back to back. The sequence numbers of the first & second segments are 230 & 290. The first segment was lost but the 2nd segment was received correctly. Let X be the amount of data carried in the first segment & Y be the ACK no. sent by the receiver.

X, Y = ?

Amount of data contained in first segment,

$$X = (290-1) - 230 + 1 = 60 \text{ bytes}$$

Ack. no = Seq. no. of 1st segment = 230

(as first segment was lost).

→ TCP congestion control.

Congestion refers to a N/W state where the traffic becomes so heavy that it slows down the N/W response time. Congestion leads to the loss of packets in transit.

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

- TCP reacts to congestion by reducing the sender window size. The size of the sender window is determined by the following 2 factors:

1. Receiver window size: It is an advertisement of how much data (in bytes) the receiver can receive without acknowledgement.

Sendee should not send data greater than receiver ws. Otherwise, it leads to dropping the TCP segments that causes TCP retransmission. So, sender should always send data less than or equal to W_R. Receiver dictates its window size to the sendee through TCP header.

2. Congestion window size (W_c):
Sender should not send data greater than congestion ws.

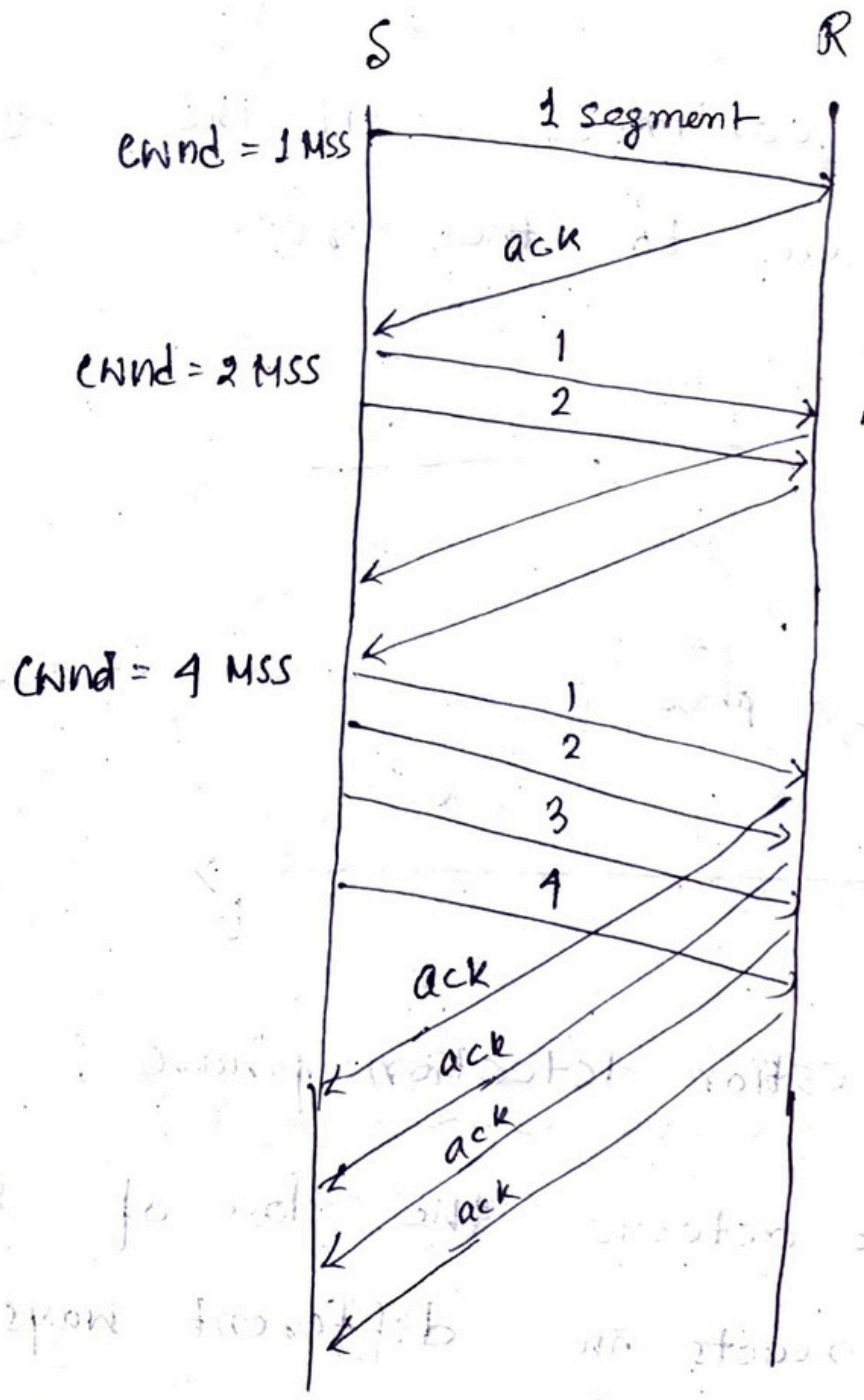
- Sendee window size =
  min (Receiver WS, Congestion WS)

- TCP congestion policy

Has 3 phases -

a) Slow start phase: Sender sets $W_c = MCax$. segment size (1 MSS). After receiving each ack., sender increases the congestion ws (W_c) by 2n MSS. In this phase size of W_c increases exponentially.

$b = 2a$.



After 1 RTT,
$W_c = 2^1 = 2$ MSS

After 2 RTT,
$W_c = 2^2$ MSS

⋮

This phase continues until the W_c reaches the slow start threshold.

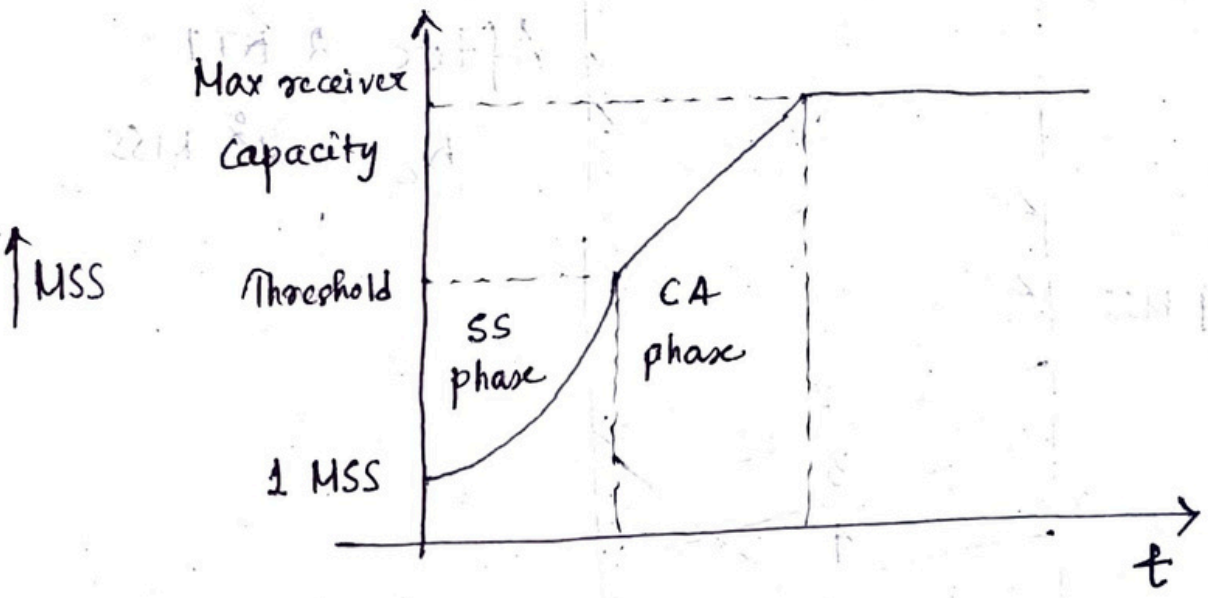
Threshold = $\left(\text{Max. no. of segments that receiver window can accomodate}\right) / 2$

$= \dfrac{\text{Receiver WS}}{\text{MSS}} \Big/ 2$

b) Congestion avoidance phase:

After reaching the threshold sender increases the W_c linearly to avoid congestion.

This phase continues until the W_c becomes equal to the receiver WS.



c) Congestion detection phase :

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

Case 1 - Detection on time out

Expiry of TO timer. Suggests stronger possibility of congestion.

In this case, sender reacts by setting the slow start threshold to half of the current W_c. Decrease the W_c to 1 MSS and resume the SS phase.

Case 2. - Detection on receiving 3 duplicate acks

Weaker possibility of congestion in the N/N. Chances that a segment has been dropped but few segments sent later may have reached.

Sender reacts by setting the SS threshold to half the current W_c. Decrease the W_c to SS threshold. Resume the CA phase.

Q. Line with 10 msec RTT & no congestion. W_R = 24 KB, MSS = 2KB. How long does it take before the first full window can be sent?

→ W_R = 24 KB = $\frac{24}{2}$ = 12 MSS

SS Threshold = $\frac{12}{2}$ = 6 ~~MSS~~ segments

SS phase: 1 $\xrightarrow{1}$ 2 $\xrightarrow{2}$ 4 $\xrightarrow{3}$ 6

CA phase: 4 → 7 $\xrightarrow{5}$ 8 $\xrightarrow{6}$ 9 $\xrightarrow{7}$ 10 $\xrightarrow{8}$ 11 $\xrightarrow{9}$ 12

9 RTTs taken before first full window sent.

9 x 10 msec = 90 msec. (Ans)

✓ * Consider an instance of TCP's additive increase multiplicative decrease (AIMD) algorithm where the W_S at the start of SS phase is 2 MSS, threshold at the start of first transmission is 8 MSS. Assume that a TO occurs during the 5th transmission. Find the W_C at the end of 10th transmission.

→ SS phase: 1 → 2 → 4 → 8 (transmission no. 1, 2, 3)

CA phase: 4 → 5 → 10 (transmission no. 4, 5)

TO

SS threshold = ~~8~~ / ~~2~~ / ~~4~~ / 7 / 4 / $\frac{10}{2}$ | $\frac{W_c \text{ (current)}}{2}$

= 5

N_c = 2 MSS (decrease W_c to 2 MSS)

resume SS phase: 6 → 7 → 8 (transmission no. 6, 7, 8)

CA phase: 9 → 10 → 11 (transmission no. 9, 10, 11)

Ans = 8 MSS.

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

$$\rightarrow \quad W_c = \frac{18\ KB}{1\ MSS} = \frac{18\ KB}{1\ KB} = 18\ MSS$$

TO {
- SS threshold = $\frac{18}{2} = 9$ MSS
- W_c = 1 MSS

SS phase. $1 \rightarrow 2 \rightarrow 4 \rightarrow 8 \rightarrow 9$.

Ans = 9 KB. (9 MSS)

✓ Q On a TCP connection, current W_c is 4 KB.

* Window advertised by the receiver is 6 KB. Last byte sent by the sender is 10240 & the last byte acknowledged by the receiver is 8192.

a) Current WS @ the sender?

b) Amount of space free in sender window?

→ a) W_s = min (4 KB, 6 KB)

= 4096 B

b) Bytes from 8193 to 10240 are still present in sender's window. Waiting for their ack. Total bytes present in sender's window

= 10240 − 8193 + 1 B

= 2048 B.

From here, amount of free space in sender's window currently

= 4096 − 2048 = 2048 B

As no ack received after 8192.

→ TCP timers. (RBR 20 TCP timer mgmt)

Used by TCP to avoid excessive delays during communication.

5 important timers ~

a) Time out timer : Used for retransmission of lost segments. Sender starts a TO timer

after transmitting a TCP segment to the receiver. If sender receives an ack before the timer goes off, it stops the timer. If sender does not receive any ack & the timer goes off, then TCP retransmission occurs. Sender retransmits the same segment & resets the timer. The value of TO timer is dynamic & changes with the amount of traffic in the N/W. TO timer is also called as retransmission timer.

b) Time wait timer: TCP uses a time wait timer during connection termination. Sender starts the TW timer after sending the ACK for the second FIN segment. It allows to resend the final ack. If it gets lost. It prevents the just closed port from reopening again quickly to some other application. It ensures all the

segments heading towards the just closed port are discarded. Value of TW timer is usually set to twice the lifetime of a TCP segment.

✓ c) Keep alive timer (Server):

Used to prevent long idle TCP connections. Each time server hears from the client, it resets the KA timer to 2 hrs. If server does not hear from the client for 2 hrs, it sends 10 probe segments to the client. These probe segments are sent at a gap of 75 secs. If server receives no response after sending 10 probe segments, it assumes that the client is down. Then server terminates the conn^{n} automatically.

Client

✓ d) Persistent timer. : To deal with a Zero-window-size deadlock situation. It keeps the window size information flowing even if the other end closes its receiver window.

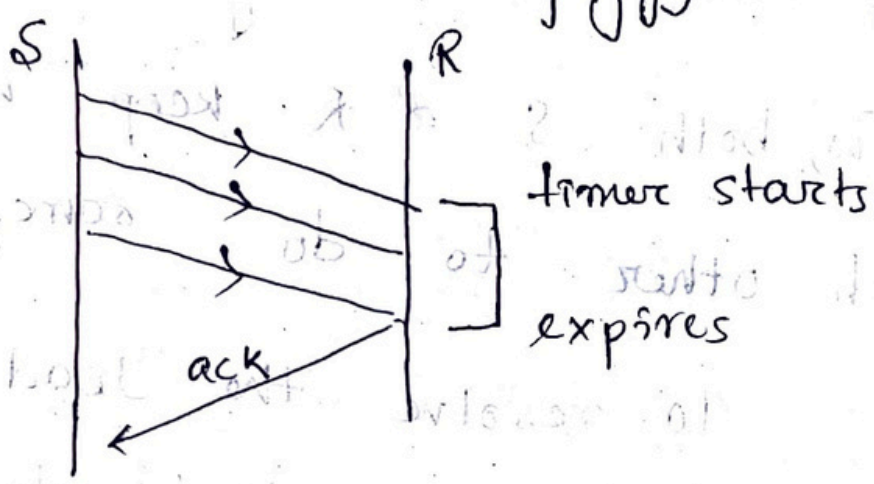
Situation - sender receives an ack from the receiver with zero window size. This indicates the sender to wait. Later, receiver updates the window size & sends the segment with the update to the sender. This segment gets lost. Now, both S & R keep waiting for each other to do something.

To resolve the deadlock, sender starts the persistent timer on receiving an ACK from the receiver with a zero window size. When persistent timer goes off, sender sends a special segment to the receiver. This is called a probe segment & contains only 1 B of new data. Response sent by the R

to the probe segment gives the updated window size. If the updated window size is non-zero, it means data can be sent now. If the updated window size is still zero, the persistent timer is set again & the cycle repeats.

e) Acknowledgement timer.

Cumulative ack & piggy backing.



- Time out timer. (21, 22, 23 rbr)

Consider receiver has sent the ack to the sender. The ack is on its way through the network.

case 1: high traffic. - If there's high traffic in the N/W, the time taken by the ack to reach the

sender will be more. So, as per the high traffic, value of TO timer should be kept large. If the value is kept small, then timer will time out soon. It causes the sender to assume that the segment is lost before reaching the receiver. However, in actual the ack is delayed due to heavy traffic. Sender keeps retransmitting the same segment. This overburdens the N/W & might lead to congestion.

case 2: low traffic - time taken by the ack to reach the sender will be less. So, as per the low traffic, the value of TO timer should be kept small. If the value is kept large, timer will not time out soon. Sender keeps waiting for the ack even when it is actually lost. This causes excessive delay.

✓ - So the value of TO timer should