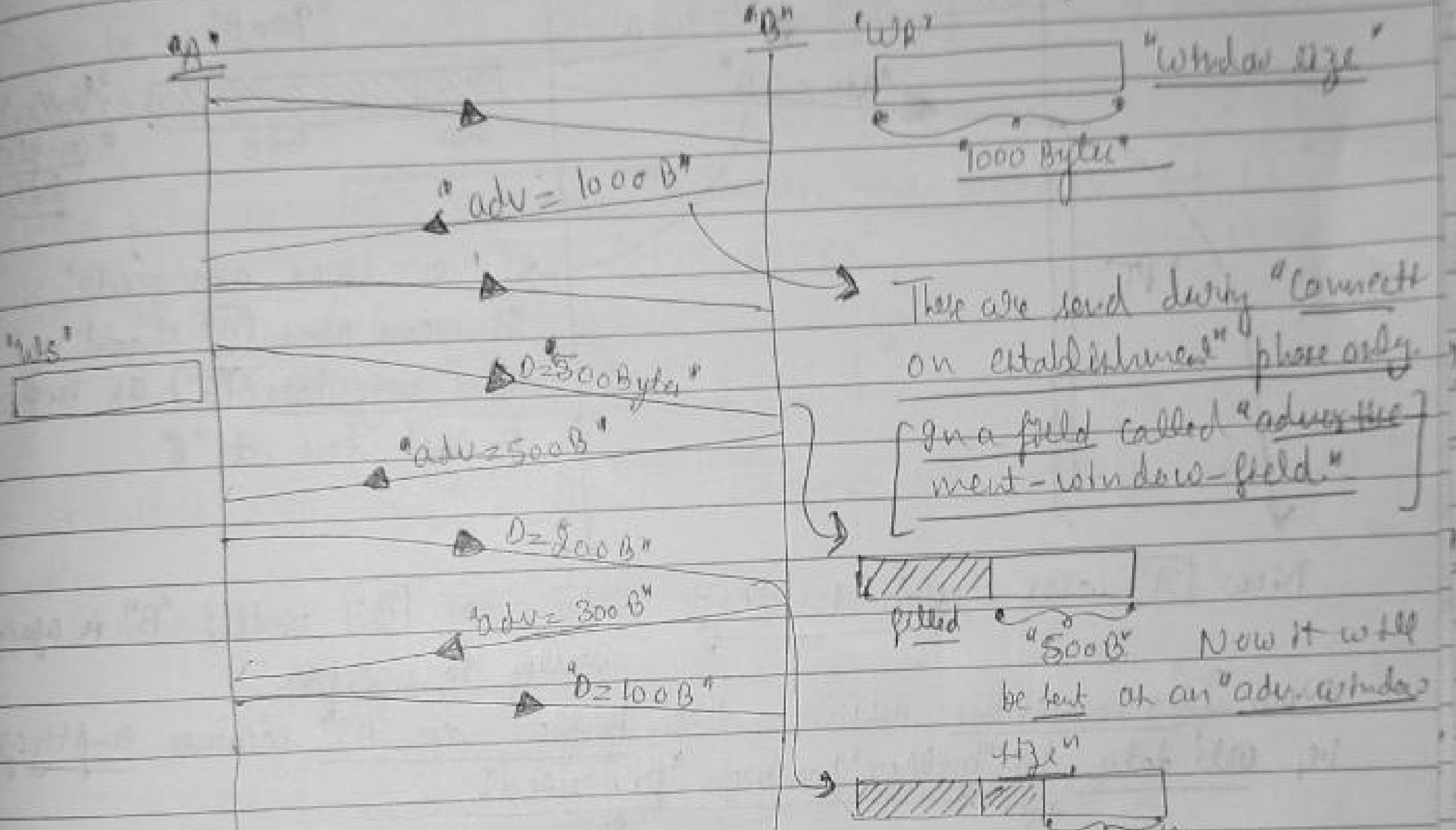
"TCP Flow Control using Advertisement Window"

Window size or adv. window field :→ "No. of bits" = $\boxed{\text{"16"}}$

It is used for "flow control" purpose

→ (A "sender" should never send more than what a "Receiver" is going to "receive")



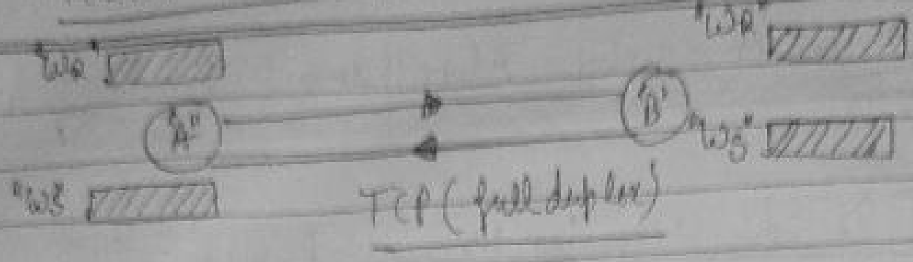
Window size of "Host 'B'" = "1000 Bytes" It means $\boxed{\text{"A"}}$ can send any no of segments, but all the "segments" put together should not cross "1000-bytes" at "receiver's side".

After receiving information about "Receiver's Window size" = "1000B"

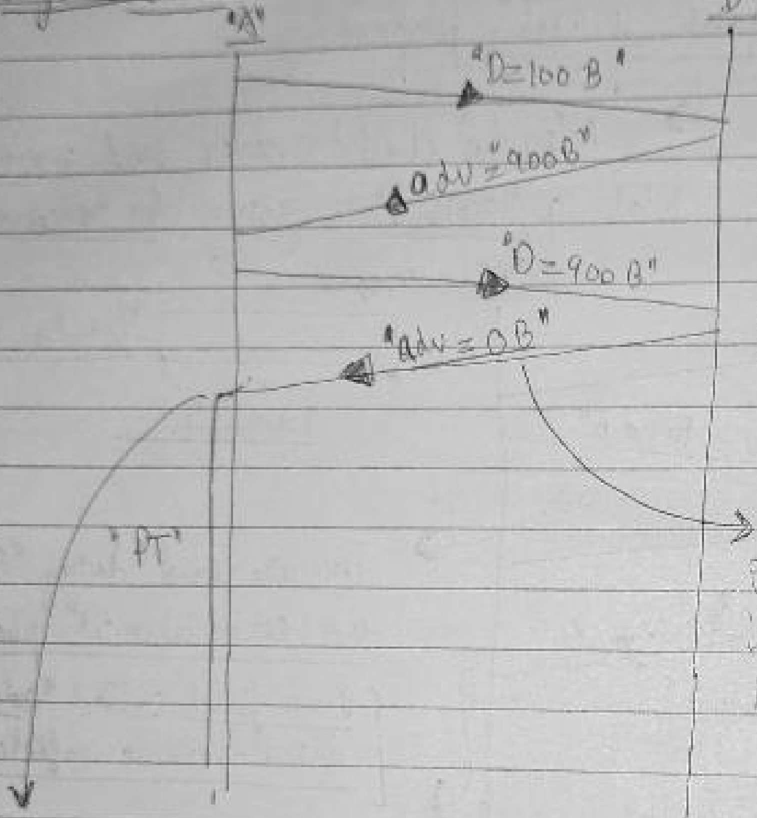
then $\boxed{\text{"A"}}$ will set "sender window size" to be "1000 B" only.

P.T.O

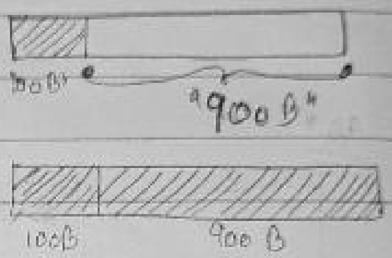
["Sender window size" is set only after "receiving "Receiver window size" through adv. window field"]



"Deadlock Condition"



By the time 100 B reach assume buffer become free

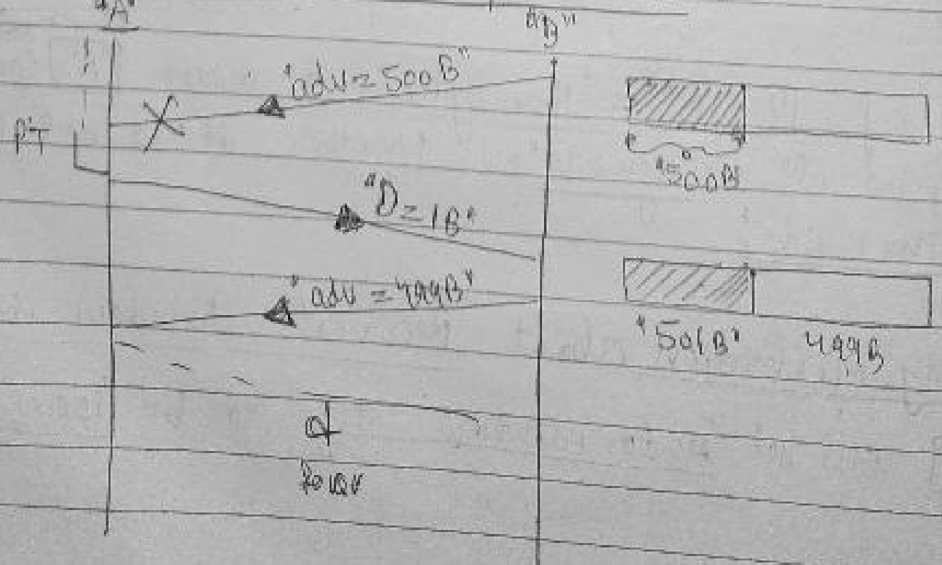


Buffer is completely full

Now since adv = "0B" it mean Now "A" should not send any data to "B" as buffer is not free at "B"

Now "A" will stop sending data to "B" until "B" is again ready to accept.

Now after sometime assume that Buffer at "B" becomes empty again i.e. all data in "buffer" has been "processed".



Now assume the packet with "adv = 500" gets lost in b/w,

then "A" will think that "B" is still busy.

&

"B" will think that "A" doesn't have any "data" to send.

This problem will actually lead to a "deadlock"

& In "worst case" it might lead to "connection-termination"

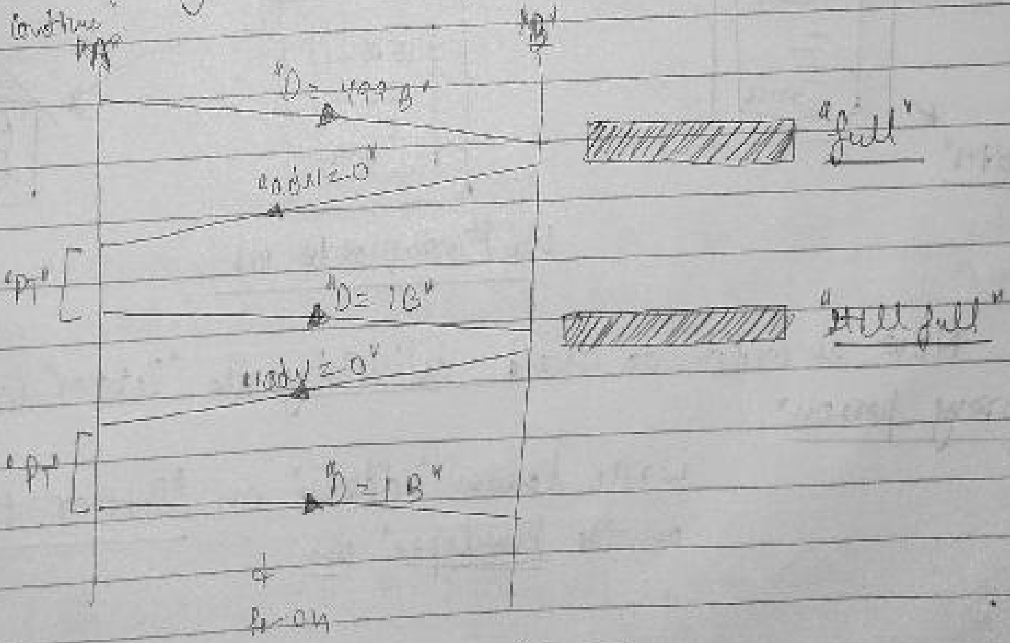
So in order to avoid this from happening "A" is having a timer called as "Persistence timer."

Now whenever "A" get an "advertisement window" as "0" "A" is going to set a "persistence timer"

"Persistent" → "Not giving up".

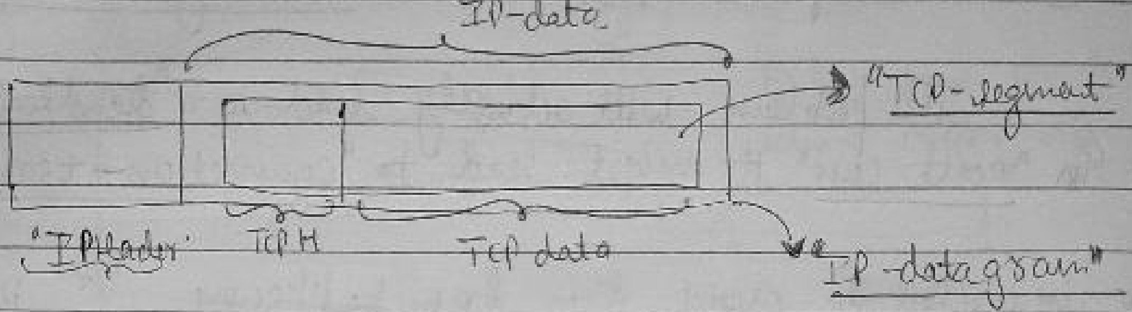
& within the time if "A" doesn't get anything from "B" side

It will test "B" by sending only "one byte" of data



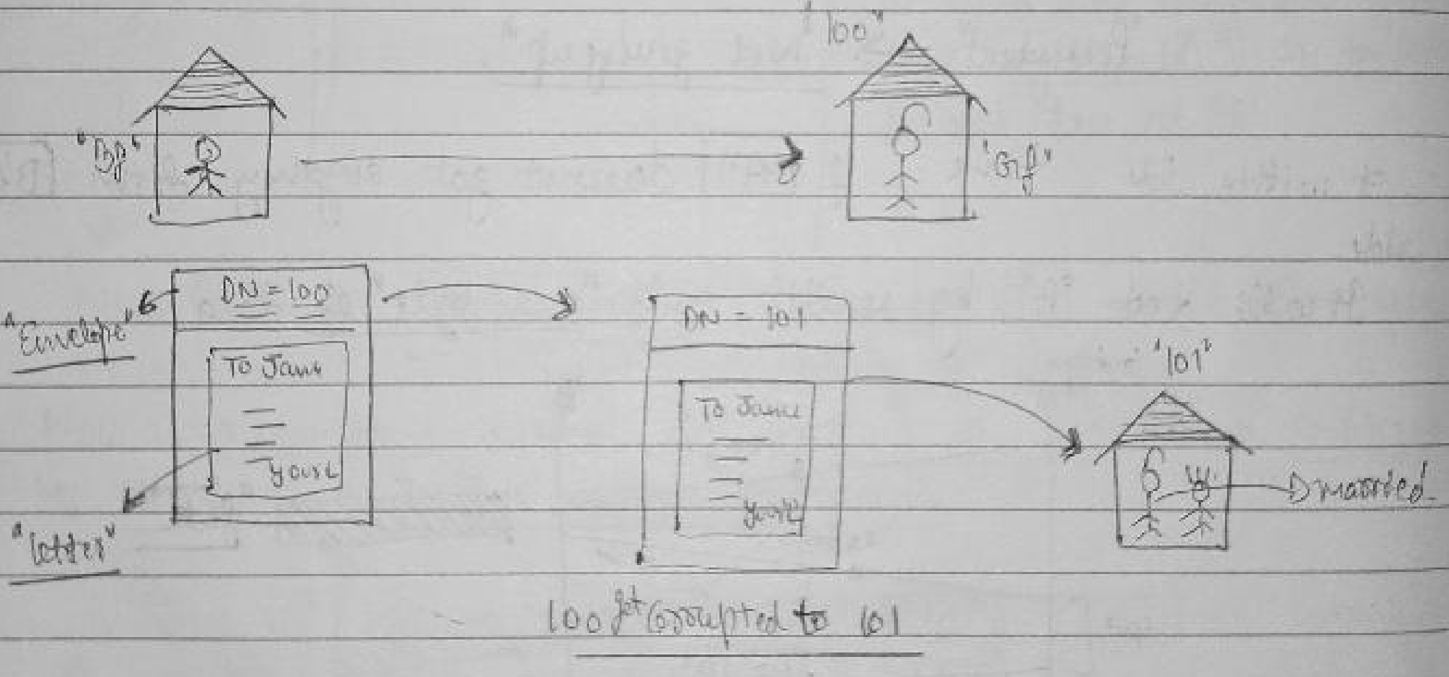
"TCP - Checksum field" ⇒ "16-bit in size"

This "TCP-Checksum" takes care of "TCP-header" as well as "TCP-data" unlike "IP header checksum" which takes care of only "IP header".



"TCP checksum" is calculated on "IP header" + "TCP H" + "TCP data".

Let us understand why "TCP checksum" is computed on "IP header" also.



100 got corrupted to 101

Now in order to avoid delivering the "letter" wrongly to a "wrong person"

write down "address" on "letter too" as well as on the "envelope" also

This "Envelope" is nothing but the "IP packet" & "TCP packet" is nothing but the "letter"

"TCP Checksum" = "TCP Header + TCP Data + IP Header"

But we are not going to include total "IP header", the reason is many of the fields in "IP-Header" will change.

So, we are going to include only important fields which are not going to change & these are:-

↳ "SIP" ↳ "DIP" ↳ "Protocol field" ↳ "TCP-segment length (TL-HL)"

"Pseudo "IP Header"

"SIP" (32) bits		
"DIP" (32) bits		
"0.0000000" (8)	"Protocol" (8)	"TCP seg. length" (16)

∴ "TCP checksum" = "TCP Header" + "TCP Data" + "Pseudo IP Header".

↓

"not actually transmitted".

Note:- "TCP checksum" is going to do double checking while "delivering packets" one at "Transport-layer" & other at "Network layer".

⇒ "Option field in TCP Header".

(0 Bytes to 40 Bytes) size.

P.T.O

"Options in "TCP" are :->

i> "Time-stamp" :-> It is used when we have to increase the "no. of sequence numbers"
(when "WAT" < "LT")

ii> "Window size Extension" :-> "window-size" = "16 bits"
↳ Which is basically "64 KB"

So we add "14-bits" to the "16-bits" & make it "30"
:-> "1 GB"

Now these "14-bits" are kept in the "Options-fields"

iii> "Parameter - Negotiation" :-> for "special purposes" while establishing "TCP-connection".
like specifying "MSS" (Max. segment size").

iv> "Padding" :-> It is used because we want "Header-size" to be a multiple of "4 always". If it is not a "multiple of '4'" we will add extra bits in the "option field" to make it "multiple of "4"".

• ——— •

→ "Retransmission-in-TCP" :-

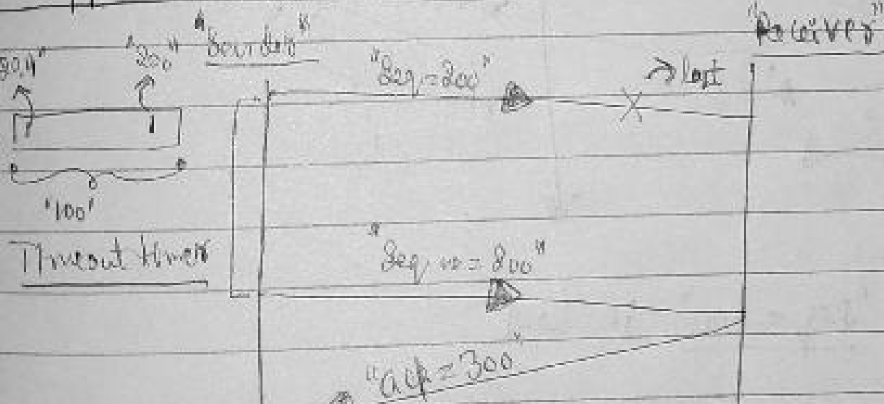
For "flow control" TCP uses a combination "SR + Go back N"

(selective Repeat) "SR" + "GBN" (Go back N)

becoz "Ws = WR"	Acknowledgement used are cumulative
Since "Ws = WR", out of order packets will be accepted at "Receiver".	
"TCP" is "75%" "SR"	& "25%" "GBN"

Now we want to discuss, what happens when a "packet is lost"? & whenever a packet is lost, we want to retransmit it & there are "various - approaches - for - retransmission"

(Retransmission after Timeout)

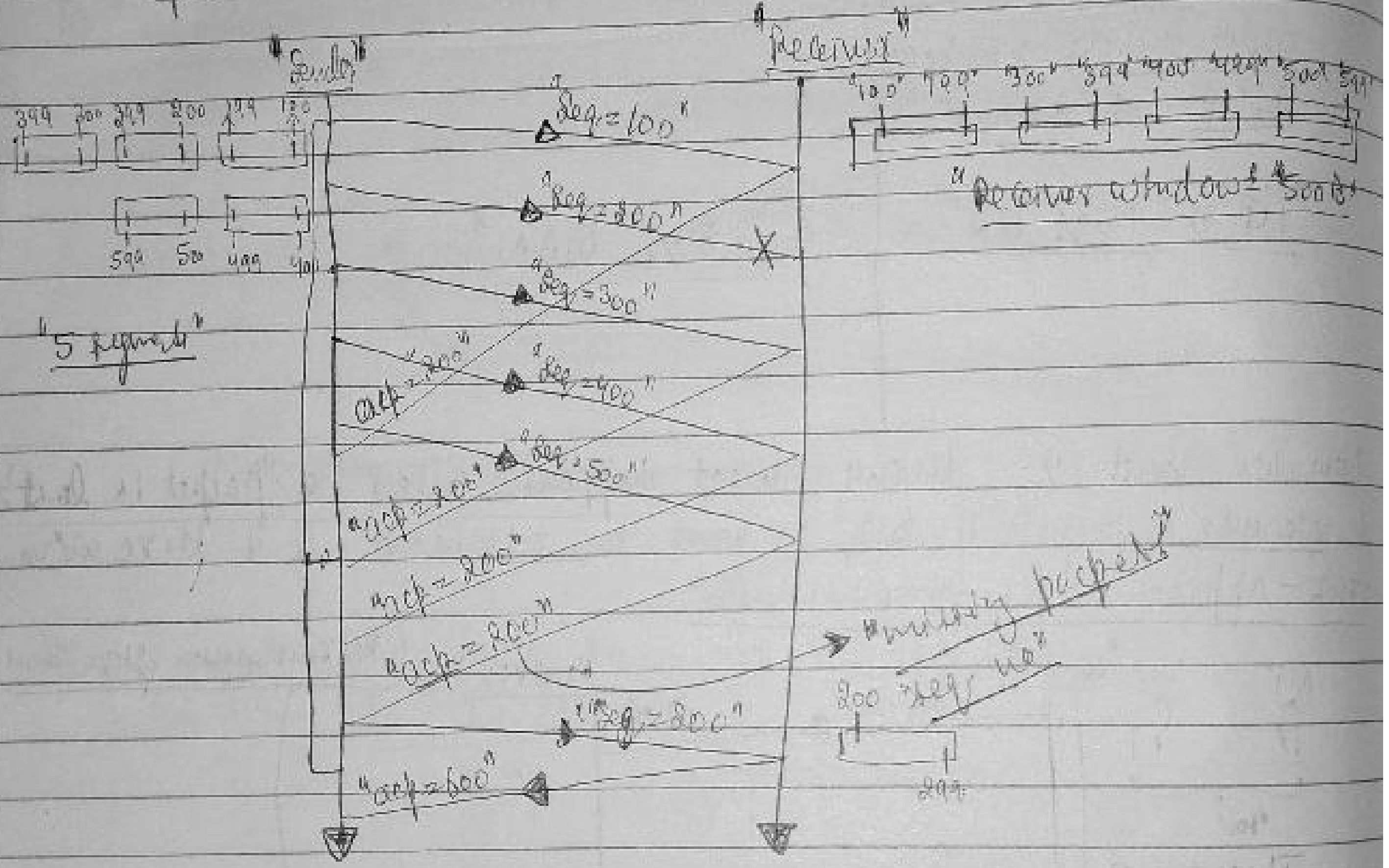


→ we are having a "timeout - timer" & once it expires, we are going to retransmit the packet again.

Along with this "TCP" follows "one more rule" :->

let us assume that "Sender" is having "five-segments" to send, with "seq no" as shown below :->

Since "TCP" follows SR protocol, out of order packets are also accepted



suppose "packet" with "seq = 200" is lost

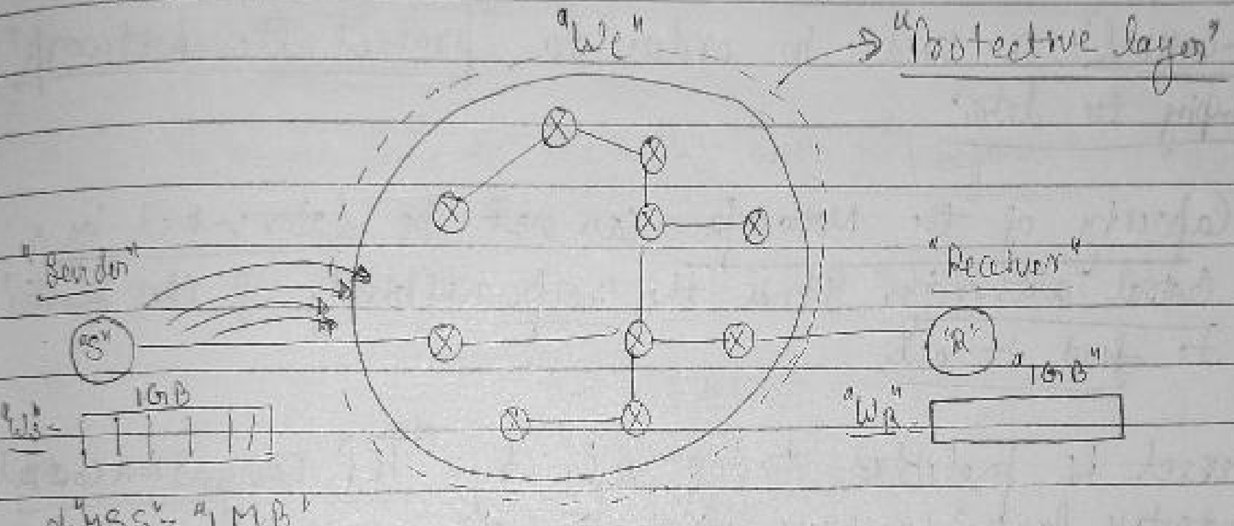
When "TCP Receiver" receives "any segment" it is not going to send the "acknowledgement" as sequence no expected next. So, instead it will send "seq no" of the segment which is missing from the earliest segment

When "Sender" receive the "duplicate acknowledgements" It will understand that, packet with seq. no 300 is missing of all the segments after this have reached successfully.

This is known as "Retransmission after 3 duplicate acknowledgements" or "Early Retransmission".

* —— *

"Introduction to TCP – Congestion – Control"



& "MSS" = "1 MB"

Now No. of segments Sender can send in one go = "1024"

Now "sender" will put all the "segments" on to the "outgoing-link" in one go, Even if "Receiver" is able to accomodate these many segments, but the underlying network is not in a position to hold "all these segments" put together

So "Sender" should not dump the "data" onto the "network" without finding out the "Capacity" of the "underlying network".

But we donot know about the "Capacity of the Network" let us assume the capacity of the network is "W_C"

P.T.O

Now a "sender" should always send a "min" of $\rightarrow$ $\boxed{\min(\text{"}W_C\text{"}, \text{"}W_R\text{"})}$

"Receiver" is protected using "flow control" by using "advertisement window field".

Now for "protecting the Network", we need "Congestion-Control":

"Congestion-Control" is used in order to "protect the network" from "dumping the data"

Capacity of the Network can not be determined by a Central authority. It is the responsibility of the "Sender" to find it out.

Our internet is protected today only by "TCP-Congestion Control". It is a "protective layer" covering the "network".

Let us now see how this "Congestion-Control - actually works":

Let us assume for simplicity that "receiver" is going to advertise some "window size" & it is going to "fix it".

Let us assume both "sender" & "receiver" have agreed upon "Window size" & "MSS" as:

Window size = "8 KB"
& MSS = "1 KB"
It means in 1 Go sender can send "8-packets"

But then "Congestion control Algorithm" says that even though you can send "8 packets" donot send them all at once becoz the underlying network op may not be in a position to handle it.

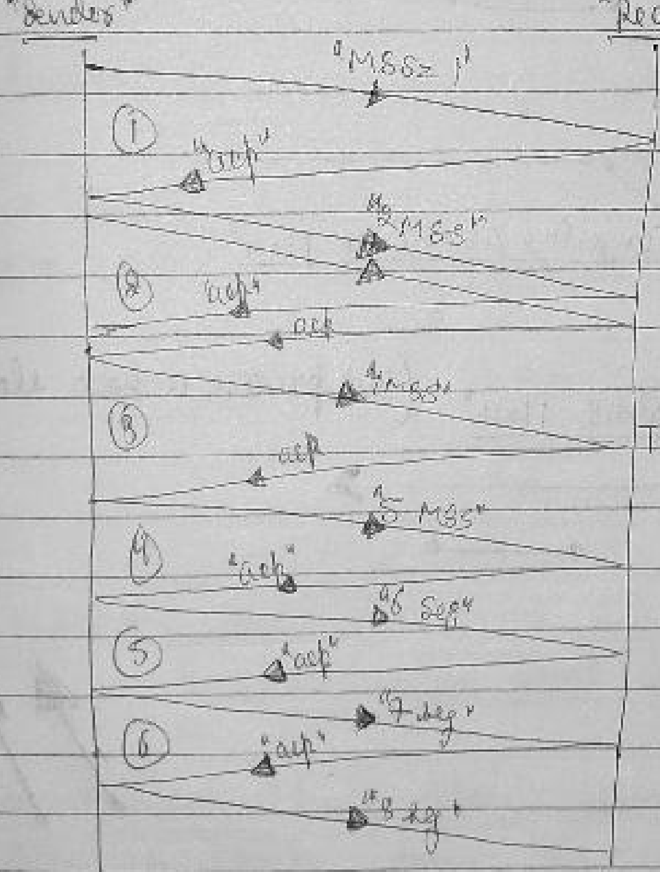
"adv wind" = "8KB"
"MSS" = "1KB"
"W_R" = "8 segments"

Here we are going to maintain one more window called "Congestion window" $\boxed{W_C}$.

& we are going to start "W_C" with = "1 segment"

Now "W_S" will be min. of ("W_C" & "W_R")

"W_S = 1 segment"



Here we are "increasing" the "no. of segments" exponentially.

Th{ So, here we first need to find the "Threshold value"

"Threshold" = "$\frac{W_R}{2}$" = "4"

So we keep on "sending the segments" exponentially till we reach "threshold value".

Once we reach "threshold" we are going to "grow linearly".

"RTT"

Seq → | 1, | 2, | 4, | 5, | 6, | 7, | 8 |

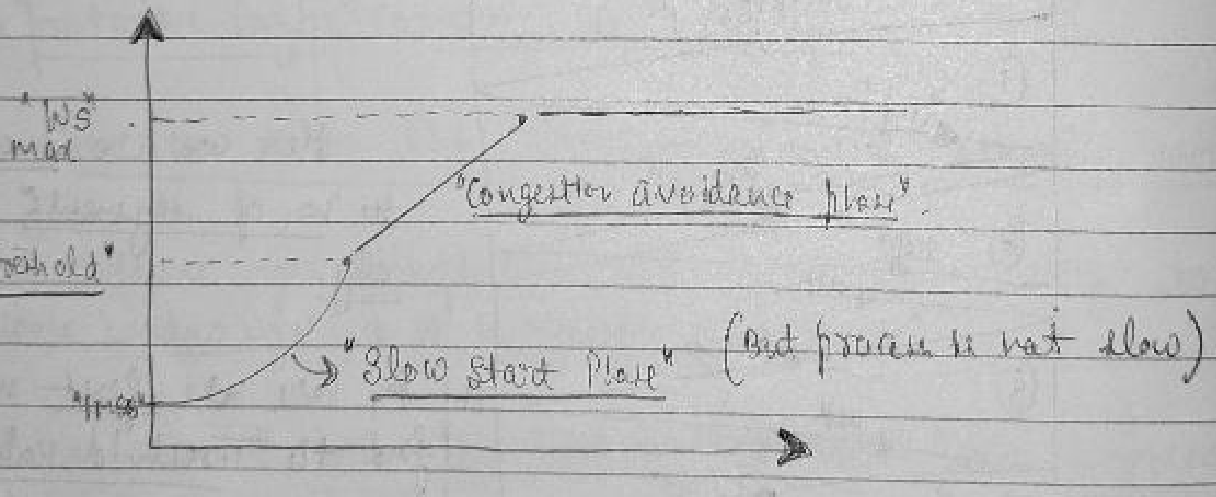
Now after "6 RTT", the "Sender window" has reached the max. capacity.

Q:→ adv. wind = "16 KB"
MSS = "1 KB"
W_R = "16 seg", $Th = \frac{W_R}{2}$ = "8"

Now find out after how many "round trip time" do we reach max sender capacity.

Sol:→ 1 | 2 | 4 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16

∴ "RTT = "11""



"TCP Congestion Control algo with an example" :→

"Congestion Control Algorithm of "TCP" has three steps"

i) 'Slow-Start Phase'

ii) "Congestion-Avoidance Phase"

iii) "Congestion Detection Phase" : (whenever a packet is lost, we can detect a congestion)

a) "Time Out" ("Severe - Congestion")
b) "Three duplicate Acknowledgements" (mild - congestion)

c) "ICMP" (Can also be used to "avoid congestion")

→ (In "ICMP" we use "Source-Quenching")

Let us understand it with the help of an "example" :→

"WR" = "64 KB" ∴ "WR" = "64 MSS"
"MSS" = "1 KB" ~~Throughput~~ / Threshold = $\frac{64}{2}$ = "32 MSS"

"Slow start phase" :→ $\underbrace{1\ 2\ 4\ 8\ 16\ 32}_{\text{"Slow-start phase"}}$

Note :→ Some slow start implementations start from "1 MSS" & some even start from "2 MSS".

"Congestion - avoidance - phase" starts after reaching the "threshold value" i.e. it starts from "33 MSS".

P.T.O

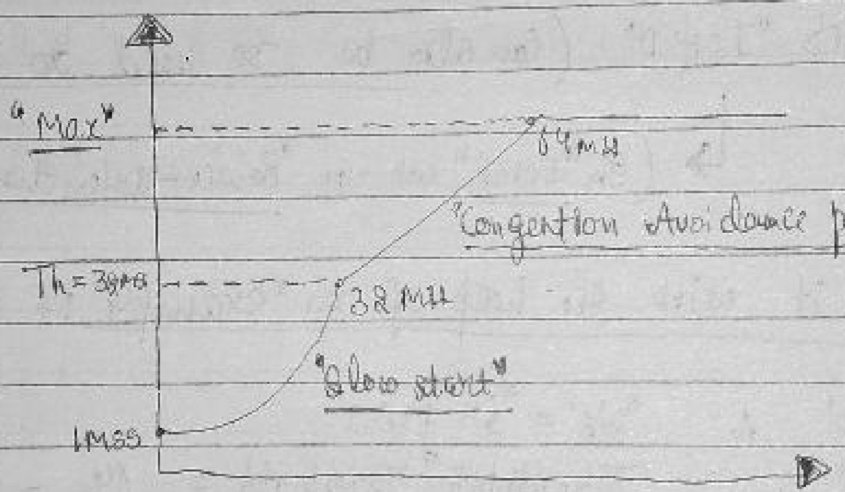
& go till "64 MSS."

"1" "2" "4" "8" "16" "32" 33 34 35 36 ---- 64 MSS

"Slow start phase" "Congestion Avoidance Phase"

Note ⇝ After "threshold", till we reach "Maximum – Receiver – Window" we will grow in a "linear fashion".

Note ⇝ There is no point in moving beyond 'Maximum – Receiver window' size'.



Now During any of these phases "Congestion" can be "detected". & we are going to "handle it".

Now let us see how "Congestion is detected" ⇝

W_R = "64 KB", MSS = "1 KB", W_R = "64 MSS".

& Threshold = $\frac{64}{2}$ = "32 MSS"

"Packets" are send as shown below :→

"1" "2" "4" "8" "16" "32" "33" "34" "35" "36" - - - - "84 MSS"

Let us suppose / assume that we have reaches ["34"] & now we are 'waiting'.

"1" "2" "4" "8" "16" "32" "33" "34"↑

It means we have sent "34 MSS" & now we are "simply waiting"

Now here "congestion" can be detected by two ways :→

↳ "Time Out". ↳ "Three duplicate Acknowledgements".

Let us suppose after "34th packet" the very first packet get lost & all the "other segments" following it also get "Lost"

∴ we get a "timeout", Now at this point the "New-threshold value" = ($\frac{1}{2}$ of current window)

∴ "New-Threshold" = "17"

Now "Algorithm again enters" ("Slow-start-Phase".)

"1", "2", "4", "8", "16", "32", "33", "34" 1, 2, 4, 8, 16, 17,

"TO" Now New TH = "17"

Now we move to "congestion Avoidance Phase"

[18, 19, 20] Now let us suppose at "20" we get "three duplicate acknowledgements".

After sending "20th packets" let us assume "next packet is lost" and all other packets after it also get lost.

Now suppose, we get "3 duplicate acknowledgements" It indicates that congestion is mild

↳ So again we have to Compute a new "Threshold value"

↳ $\left(\frac{20}{2}\right)$ = "10 MSS"

Here we enter "Congestion Avoidance phase".

It means we start from New Threshold of "10" & after it we grow linearly.

(18 19 20 , 10, 11, 12, - - -)

There are two ways to detect congestion :→

1> If we detect it using "Time out-method" then "New Threshold value" = ($\frac{1}{2}$ of current window) & Algorithm enters "Slow start phase".

2> In case "Congestion" is detected using "Three duplicate Acknowledgement-method":

Then algorithm enters "Congestion-Avoidance Phase".

"3-duplicate acknowledgements" method means that there is mild congestion here.

"Th = 6"   "Th = 4"

18, 19, 20, 10, 11, 12 "To" 1, 2, 4, 6, 7, 8, 4, 5, 6, 7, 8 --- 64

"3dA"

"TCP- Timer - Management" :→

i. "Time-Wait - Timer".
ii. "Keep - alive - Timer."
iii. "Persistent - Timer."
iv. "Acknowledgement - Timer."
v. "Time-Out - Timer."

"Popular timers"

i. "Time-Wait -Timer" :→ whenever we get a request to close the "connection", we donot close it immediately but instead we wait for sometime.

Eg :→ let us say we have a "process-A" which is having a connection open with "process-B".

If immediately "B" closes the connection, then what happens is port no. of "B" becomes available for some other "process".

So, "B" will not immediately release its "ports" instead it will wait for sometime before releasing the "port"

↓ This "time" is known as "Time - wait - Timer"

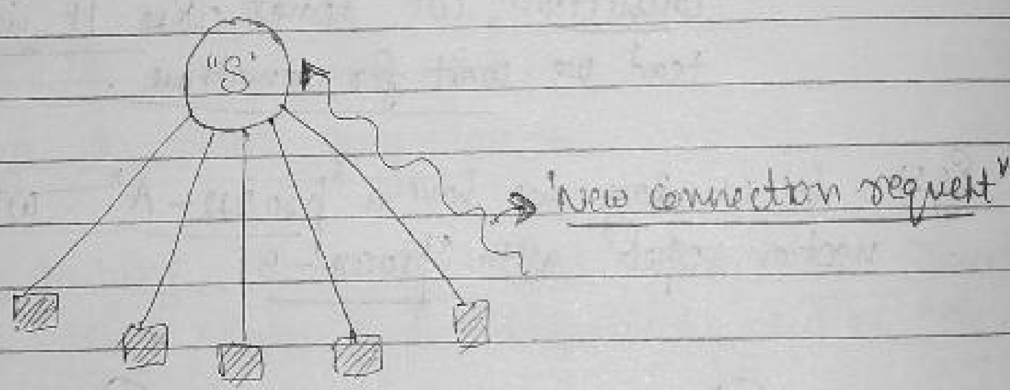
P.T.O.

Time wait - Timer = $\boxed{\text{"2 * LT"}}$

The reason is a process might accept a packet meant for a previous process', with whom 'connection' has been terminated immediately. For this reason, we donot immediately release the ports after 'connection-termination"

(ii) "Keep-alive Timer":→ "Keep alive timer" is used to keep track of "idle-connections".

Let us assume that there is a "server" & there are many connections are opened by "many clients" & the "server" is very very busy & all its resources are given away



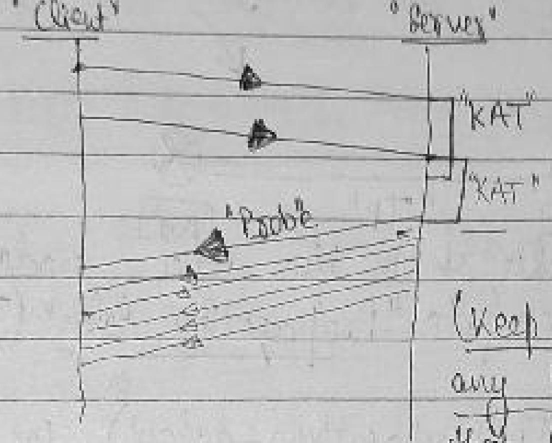
Now if a "new connection-request" arrives, 'server" is not able to fulfill it.

But there is a problem with this & it is, the "clients" might not be using the "connections. They have opened up a connection but are not sending any data". & are not involved in any "activity still holding up the resources. (which is a mere wastage of resources)

So what the "server" should do is, It should periodically check for "connections" & will terminate the connections which are ideal for long time.

Main advantage of "Keep alive Timer" is to close the "Ideal connection"

Note :> "Keep alive timer" is going to remember, when for the last time we have heard something from a "station".



Now within this "KAT" (Keep alive time) if we donot get any packet then we can assume that the client is idle & we can terminate the "connection".

"Probe" msg is sent from "Server" to the "client" at the end of "keep-alive Time".

When within "KAT" if we donot receive any 'packet' from the client, then at the end of this "KAT", we are going to send a "probe request" to the "client". Now to this "probe message" client should reply.

If reply to "probe request" doesn't arrive then "server" can think that "client" is "ideal" & it will "terminate the connection".

"Server" will send "multiple probe requests" if client doesn't respond to any of the probes, then "server" will terminate the "connection".

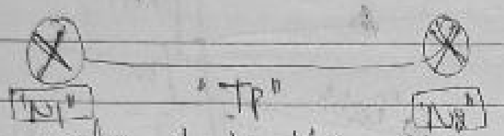
iii) "Persistent-Timer" :> Used whenever "Rcv. window = 0" is advertised.

iv) "Acknowledgement-Timer" :> "Acknowledgement timer" is used to send cumulative acknowledgements to many packets at once.

It is also used to "Implement" "Piggybacking Acknowledgement".

(v) "Time out-Timer" :-> "Timeout timer" management is very difficult at TCP (Transport layer) than "Datalink layer".

In "DLL", it deals with only "Hop-to-Hop" connection



There are two nodes & in b/w the nodes there is a "link" & it is very easy to compute "Propagation delay" (Tp) for this link

Now using the "Tp" ("Propagation delay") we can easily calculate "Round-Trip-Time" & $\boxed{\text{"RTT"} = \text{"}2*T_P\text{"}}$

& "Timeout" can easily be found as :->

$\boxed{\text{"}T_O = 2*RTT\text{"}}$ "2*RTT"
↳ "Round trip time"

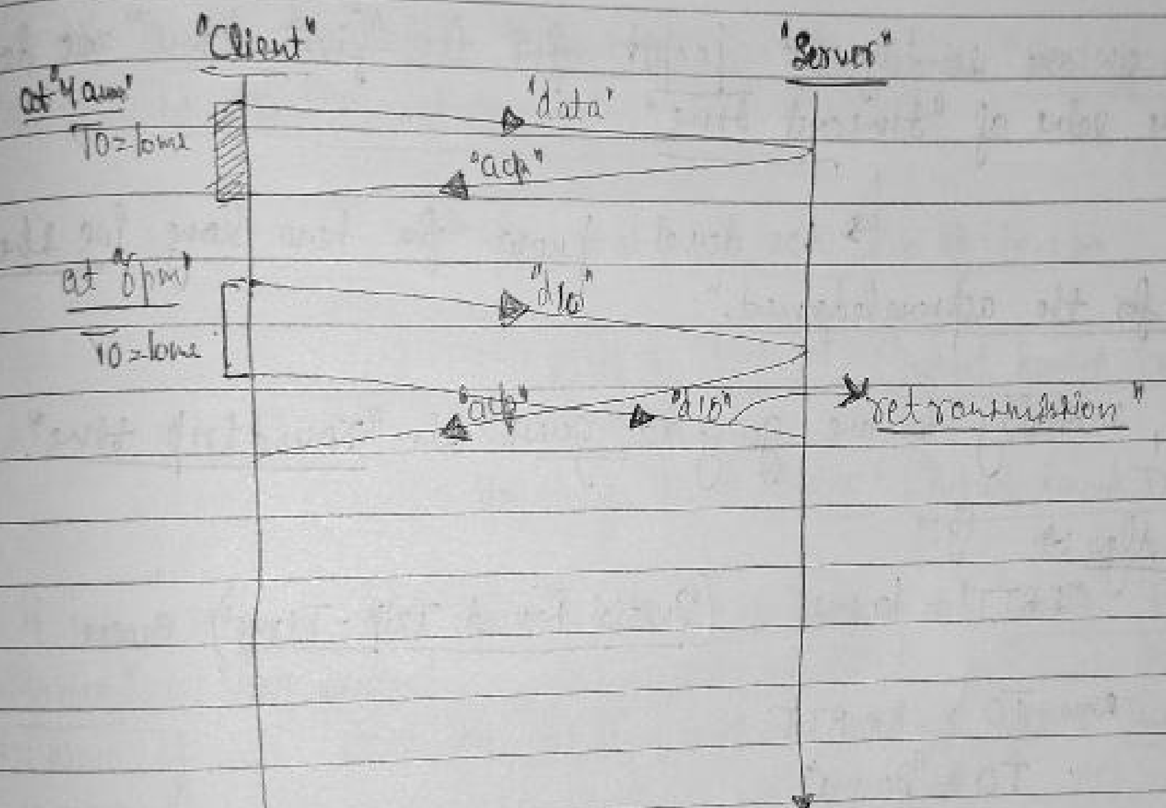
Whereas in case of "Transport layer" :-> there can be a number of routers" b/w a "sender" and a "receiver". & we donot know about the "propagation delays" b/w the 'Hops'

Here even if by some means we are able to find out propagation delay for a "link", Then it doesn't many we would be keeping this "propagation delay" as "constant".

Hence it is very difficult to compute "Time-out time" becz we are unable to predict "Propagation-delay" at "TCP" (Transport layer).

i.e. at "TCP" it is very "difficult to compute" "Time out time" for this

∴ We need "dynamic Time out timer".



At "4am" suppose we send a "packet" & set "Time-out timer" to be "10 ms". Since traffic is less, so within this "10 ms" we receive the "acknowledgement" within the "Time out"

Now suppose at "6 pm" we again send a "packet". Now since due to "heavy traffic" we will not receive the "acknowledgement" within the time out originally specified at "4 am". So, even if the "packet" is not lost we are unnecessarily retransmitting the "data-packets". assuming that the packet has lost but instead it hasn't lost

Since there is "no-congestion" but due to "unnecessarily - retransmitting" the packets, which may lead to congestion in the network.

Note:→ Now instead of retransmitting the packets, as traffic increases gradually, we should increase the "time out timers" gradually rather than retransmitting the packets.

—

⇨ "Basic - Algorithm for Timeout Timer Computation" :→

Whenever we are sending the packet for the "first time" we donot know the value of "timeout time"

↳ we donot know for how long "we should wait for the acknowledgement."

So, initially we are going to guess the "round trip time".

Basic Algo :→ Eg :→

"IRTT" = 10 ms ("Initial Round Trip-Time") Given

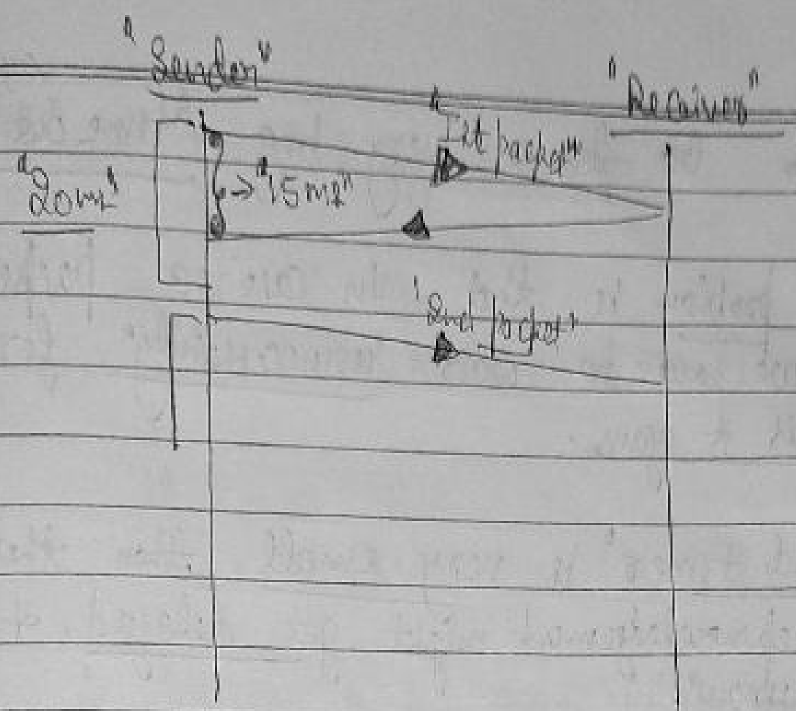
Now TO = 2 * RTT

∴ TO = "20 ms"

So we are sending a packet & we are expecting the "acknowledgement" to come back in "20 ms".

But now assume that the packet came back in "15 ms"

So ARTT = '15 ms'



So, if this is the case then when we are sending the "next packet", then what will be the "timeout timer"

→ for that we have two options :→

1> Stick to "IRTT" (Initial Round Trip Time)

or

2> change it to "ARTT" (Actual Round Trip-Time)

If we stick on to "IRTT" then it is known as "Static Time out timer". (Very static)
& Now if we stick to "ARTT" (whatever we get after "initial packet" is received.)

then we are "dynamic" (very dynamic)

Both of these ("very-static" & "very dynamic") are "very dangerous".

Because "traffic might change". (in case of dynamic & static)

"P.T.O"

"Other solution is": Go for a very big "time out timer"..

But the problem is that in case a packet has lost then still we have to wait "unnecessarily" for a "long time" to retransmit it again.

& if "timeout timer" is very small then there is a possibility that acknowledgement might get delayed, & hence unnecessary "retransmissions".

If "TO ↑" ("Timeout timer" is too large)
then it leads to unnecessary "time wastage"
(in case packet is lost)..

& if "TO ↓" ("Timeout timer" is "too small")

then it leads to unnecessary "retransmissions"
(in case we donot wait for "acknowledgement" for good enough time)

Now let us see how to choose an 'appropriate' timeout timer':

Eg:→ "IRTT" = 10 ms (assumed) Given

TO = 2×RTT
= 20 ms

& Actual RTT = "15ms"

Now let us compute "New RTT" for a new packet

$$\boxed{NRTT = \alpha * IRTT + (1-\alpha)\, ARTT}$$

where "α" is Smoothening factor.

if α = "1" then

$$[NRTT = IRTT]$$

&

If $\alpha = 0$ then

$$[NRTT = ARTT].$$

∴ "α" has to be some value b/w "0" & "1".

i.e $\boxed{0 \leq \alpha \leq 1}$ → will be given in "exam"

Now for this example let us assume that α = "0.5".

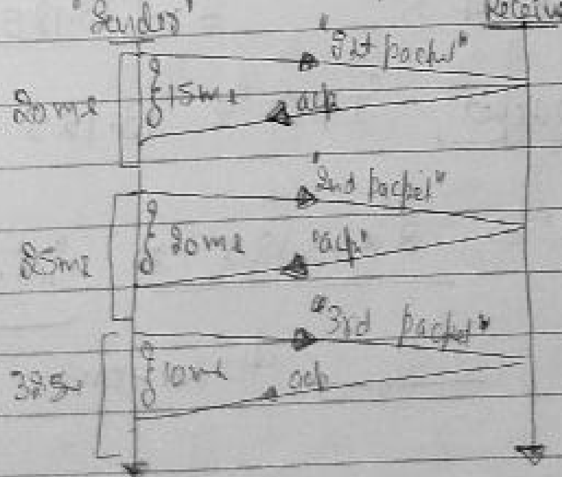
$$\therefore NRTT = \alpha(IRTT) + (1-\alpha)(ARTT)$$

$$= (0.5)(10) + (0.5)(15)$$

$$= 5 + 7.5 = \text{'12.5'}$$

∴ Next 'RTT' = '12.5 ms'.

∴ "TO" for Next packet = "25 ms"



We assume that '2nd - packet's' ack will come back in '25ms' but actually it came back in '20ms'.

∴ for 2nd packet ARTT = '20 ms'

Now depending on "ARTT" & "IRTT" for "2nd packet" we have to compute 'Next RTT' for 'third - packet'

$$\therefore NRTT = \alpha(IRTT) + (1+\alpha)ARTT$$

$$= (0.5)(12.5) + (0.5)(20)$$

$$= 6.25 + 10 = \text{"16.25 ms"}$$

∴ for Next packet RTT = "16.25 ms"

∴ "IRTT" for "third packet" = "16.25 ms"

& 'TO' = 16.25 * 2 = "32.5 ms"

∴ for "3rd packet", "ARTT" = "10ms"

Now NRTT for 4th packet =

$$NRTT = \alpha(IRTT) + (1-\alpha)ARTT$$

$$= 0.5(16.25) + 0.5(10)$$

$$= 8.125 + 5 = \boxed{13.125 \text{ ms}}$$

Now for 4th packet, "IRTT" = 13.125 ms

& ∴ "TO" = 2 * 13.125 = "26.25 ms"

—o— & so-on

"JACOBSON's Algorithm to set the Timeout-Timer"

Eg:>

Initially we don't know about the "Timeout timer". So we guess the "initial Round Trip time".

Eg ①:> let "IRTT" = "10 ms" & along with this we guess that initial deviation is

ID = "5 ms" (there could be a deviation in "10 ms" "RTT" by a factor of ± 5 ms)

∴ "IRTT" = (5 to 15) ms. range.

Now for calculating "timeout" "Jacobson" has derived a formula:>

$$\boxed{TO = 4 * D + RTT}$$

D → "Deviation"; RTT → "Round trip time"

∴ TO = 4 * 5 + 10
= 30 ms.

But ARTT = "20 ms"

Now Actual deviation AD = |(IRTT − ARTT)|

⇒ AD = |(10 − 20)|

⇒ AD = 10 ms

let $\alpha = 0.5$

Now NRTT = α(IRTT) + $(1-\alpha)$(ARTT)

= 0.5(10) + (0.5)(20)

= 5 + 10 = 15 ms

8. New Deviation "ND" $= \alpha(ID) + (1-\alpha)(AD)$

$= 0.5(5) + (0.5)(100)$

$= 2.5 + 50 =$ "7.5 ms"

Now for second packet :-

$IRTT = 15$ ms

$ID = 7.5$ ms

$TO = 4 * D + IRTT$

$= 4 * 7.5 + 15$

$=$ "45 ms"

Let $ARTT = 30$ ms (Actually happened).

$\therefore AD = 15$ ms $(15 - 30)$

$= 15$ ms

Now "NRTT" $= \alpha(IRTT) + (1-\alpha)(ARTT)$

$= 0.5(15) + (0.5)(30)$

$= 7.5 + 15 =$ "22.5 ms"

Now $ND = \alpha(ID) + (1-\alpha)(AD)$

$= 0.5(7.5) + (0.5)(15)$

$= 3.75 + 7.5 =$ "11.25 ms"

"Now for third packet":

$$\text{"IRTT"} = 22.5 \text{ ms}$$
$$\text{"ID"} = 11.25 \text{ ms}$$

$$\text{"TO"} = 4*D + RTT$$
$$= 4*11.25 + 22.5$$
$$= 45 + 22.5 = \text{"67.5 ms"}$$

Now $ARTT = \text{"10 ms"}$

$\therefore$ "AD" $= 22.5 - 10 = \text{"12.5 ms"}$.

Now "NRTT" $= \alpha(IRTT) + (1-\alpha)ARTT$

$$= 0.5(22.5) + (0.5)(10)$$
$$= 11.25 + 5 = 16.25 \text{ ms}$$

Now $ND = \alpha(ID) + (1-\alpha)(AD)$

$$= (0.5)(11.25) + (0.5)(12.5)$$
$$= \text{"5.625"} + \text{"6.25"} = \text{"11.875 ms"}.$$

Now TO for new packet $= 4*D + RTT$
$= (63.75)$ ms

This is how "Jacobson Algorithm works".

Q:→ IRTT = "10 ms" ID = "5 ms", α = "(0.5)" Successive acknowledgement for next three packets arrived in 20 ms, 30 ms, 10 ms respectively then what is the timeout for the fourth packet.?

Sol:→ "1st" IRTT = 10 ms

ID = 5 ms

ARTT = 20 ms

$\therefore AD = (IRTT - ARTT) = (10 - 20) = 10 ms$

Now $NRTT = \alpha(IRTT) + (1-\alpha)(ARTT)$

$= (0.5)(10) + (0.5)(20) = 5 + 10 = 15 ms$

Now $ND = \alpha(ID) + (1-\alpha)(AD)$

$= (0.5)(5) + (0.5)(10) = 2.5 + 5 = 7.5 ms$

Now for 2nd packet:→

IRTT = 15 ms

ID = 7.5 ms

ARTT = 30 ms

$\therefore AD = |IRTT - ARTT| = 15 ms$

Now $NRTT = \alpha(IRTT) + (1-\alpha)(ARTT)$

$= (0.5)(15) + (0.5)(30)$

$= 7.5 + 15 = 22.5 ms$

Now $ND = \alpha(ID) + (1-\alpha)(AD)$

$= 0.5(7.5) + (0.5)(15) = 3.75 + 7.5 = 11.25 ms$

for Third packet :-

$IRTT = "22.5\ ms"$

$ID = "11.25\ ms"$

$ARTT = "10\ ms"$

$\therefore AD = |IRTT - ARTT| = "12.5\ ms"$

Now $NRTT = \alpha(IRTT) + (1-\alpha)(ARTT)$

$= (0.5)(22.5) + (0.5)(10)$

$= 11.25 + 5 = "16.25\ ms"$.

Now $ND = \alpha(ID) + (1-\alpha)(AD)$

$= (0.5)(11.25) + (0.5)(12.5)$

$= 5.625 + 6.25 = "11.875\ ms"$

for fourth packet :- $IRTT = 16.25$, $ID = 11.875$

$\therefore TO = 4 * D + RTT$

$= 4 * 11.875 + 16.25$

$= 47.5 + 16.25 = \boxed{63.75\ ms}$

Ans

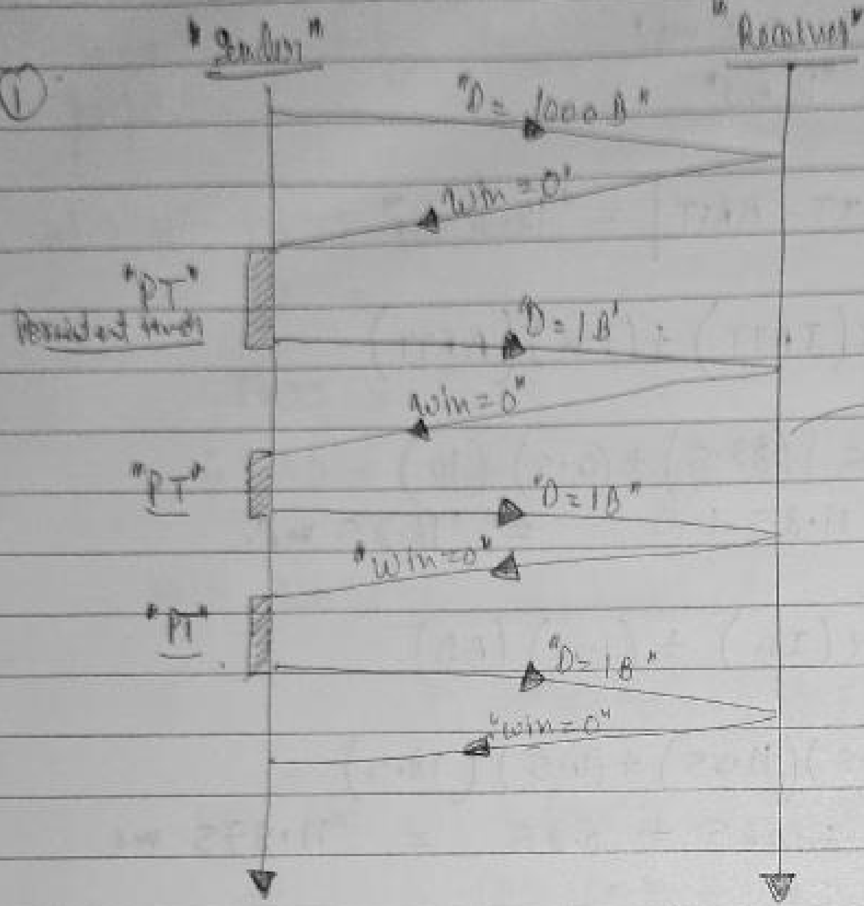
—— 0 ——

P.T.O

Date / /
Page

"Silly Window Syndrome" → It occurs in various cases. One case is like :-

1



"It is full"
WR = 1000 B"

It is a transient problem it gets fixed with some time

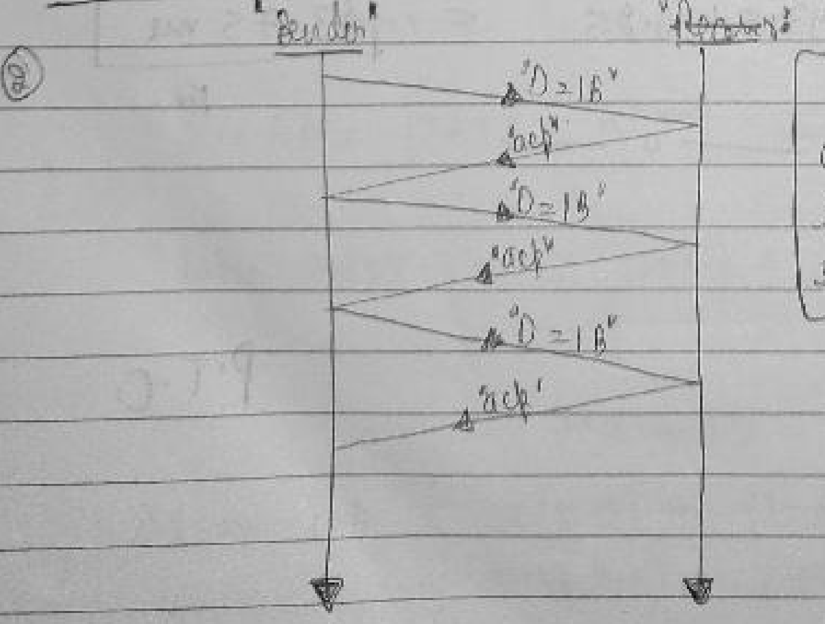
This is known as "Silly window syndrome".

We keep on sending "1 B" & it keeps on sending "0" as advertisement window.

If it keeps on "happening like this" then this is known as "silly window syndrome."

We are not doing any "useful work", we send $\boxed{\text{"1 B"}}$ & "Receiver will discard it" & send $\boxed{\text{'0'}}$ as an "advertisement window".

2

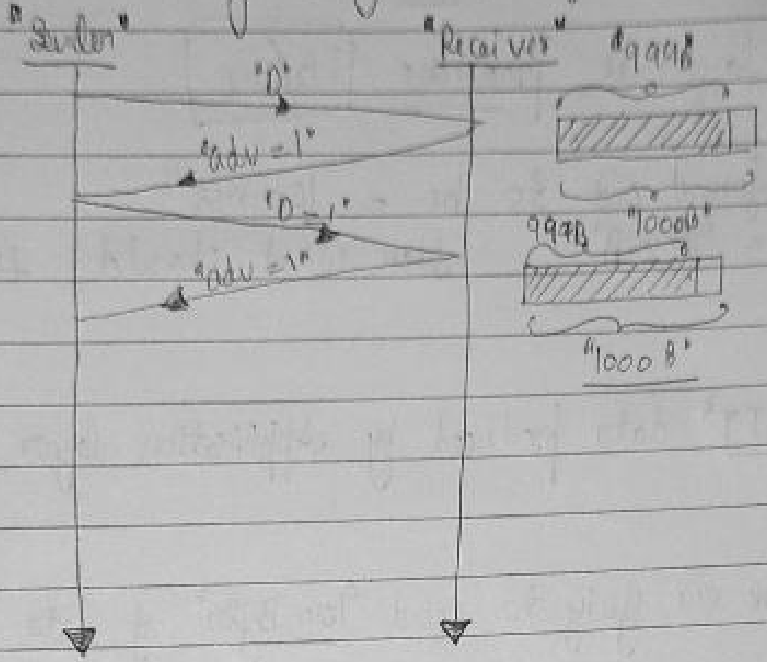


Sender is sending very small amount of data & receiver is sending acknowledgement for these small amounts.

"It is also a "silly window""

This problem (8) arises when "sender" is producing very little amount of "data" everytime.

(9) "Receiver is consuming only "one byte" at a time"



(9) is not a "transient problem" but infact it is "persistent in nature" & it will lead to inefficiency.

In order to solve "problem (8)" where "sender" is producing only "one byte" of data each time.
We have an algo called "Nagles Algo"

It says that if "sender" is very slow in producing "data bytes", then the "transport layer" should not send "1B" of data, but the "transport layer" should wait for "1 RTT"

Initially we send $\boxed{\text{"1B"}}$ & by the time "acknowledgement comes back" then we see in this amount of time, How much is the data, that we can buffer?

let us say in "1RTT" we buffer $\boxed{\text{"50 B"}}$ of data

PTO

then in next packet send "50 B" instead of "1 B"

Now, In case "data buffered" is greater than "MSS" then send exactly "1 MSS" of data.

Q:→ "Application layer" is producing 1 B/ms.

& "RTT" is found out to be = "100 ms"
& "MSS" = "200 B" then what should senders transport layer do?

Sol:→ In "1 RTT" data produced by Application layer = 100 * 1
= '100 Bytes'

∴ we are going to send "100 Bytes" of data in one segment.

Note:→ "Nagles algorithm" is going to help us whenever Sender is very slow.

When "destination" is consuming only "one byte" at a time then to handle it "Clark" has proposed a solution.

"Destination" should not "advertise 1 Byte window". It should wait for "1 MSS window".

or at least $\boxed{\text{"}\frac{1}{2}\text{ of buffer"}}$

— o —

"Go to Next NoteBook"