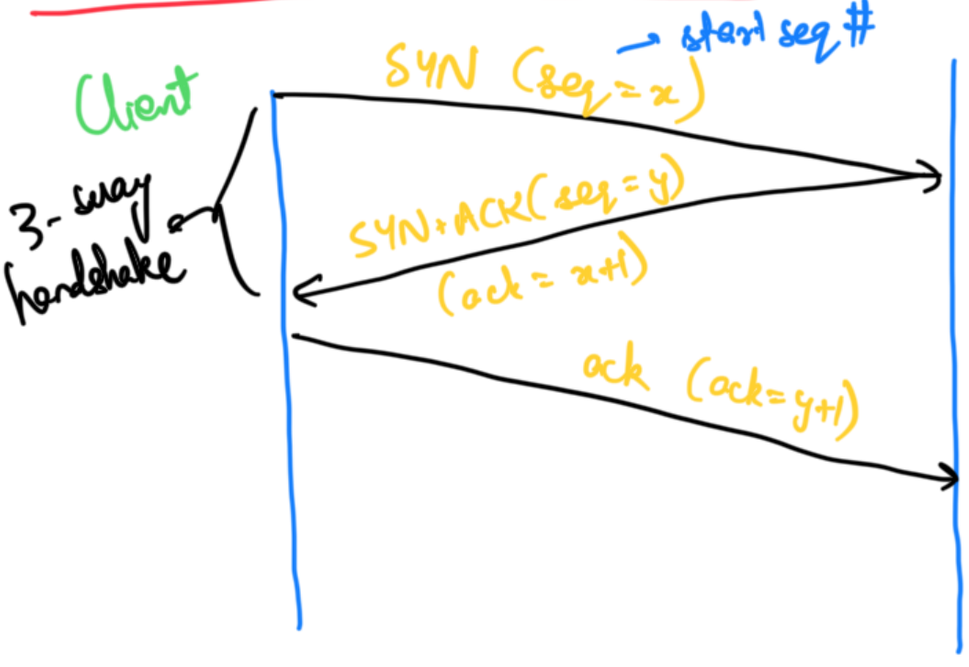
CS348 Notes
TCP
Video Numbers: 23, 24, 25

OjMaha

I have prepared these notes by watching the videos from Networks Playlist. The following notes may be asynchronous and irrelevant to what Prof. Vinay teaches in class (cuz I do not pay attention during lectures lol). Further, these notes might not cover *everything* as explained in the video lectures. Consider these to be a supplemental read :). If you find any errors, do notify me so they can be edited.

TCP (it is a duplex connection)

Connection Establishment:



SYN+ACK means both flags are set.

- SYN & FIN segments do not have data but considered to communicate 1 byte. (TCP header + 1 byte)

Here, the data stream is org from client to server.

Also, note that x, y need not be 0. It is chosen randomly.

SYN

ACK # x+1 recd by client means that the server got everything from the start till x.

Why $x, y \neq 0$? Why are they chosen randomly?

Say there are 2 streams that client wants to share with server one after the other on the same port. Suppose client has shared all segments from the first stream and starts to send segments from second stream. But say there

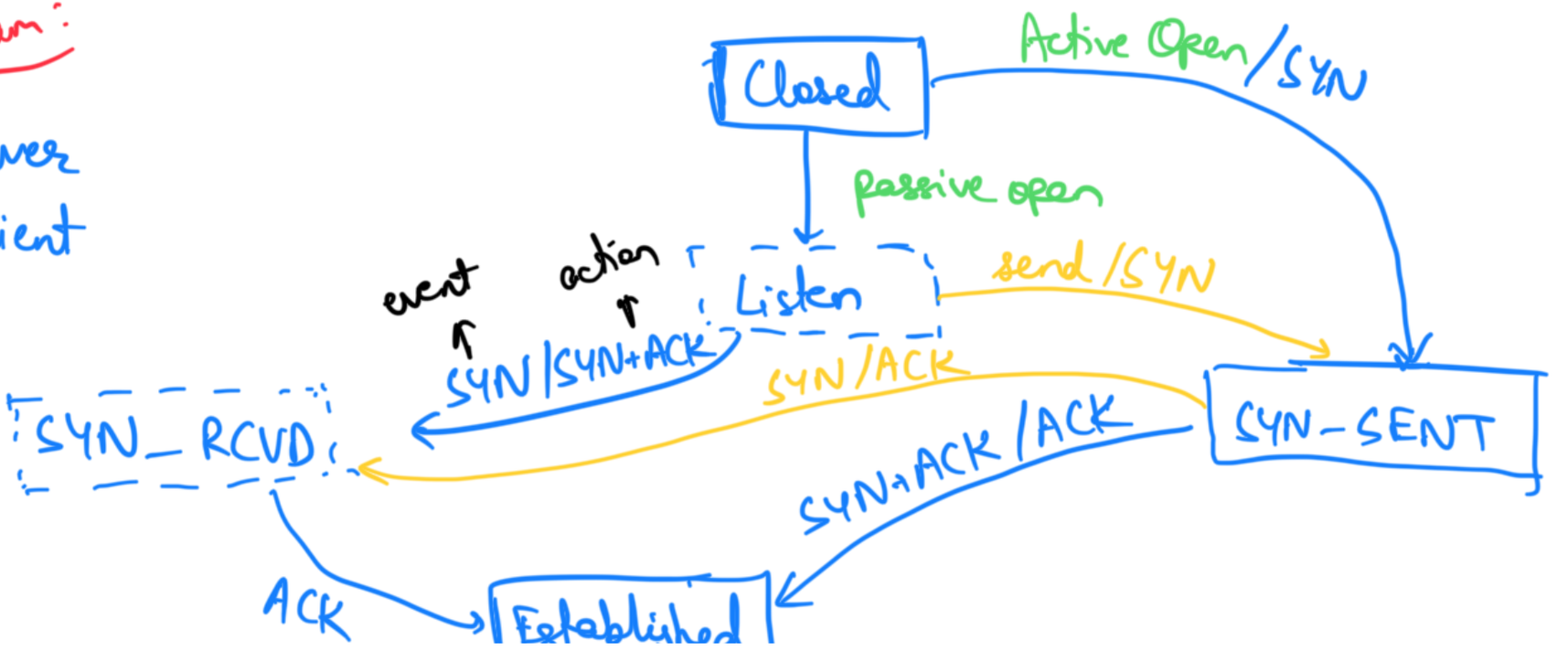
is some particular segment (#N) that is still lurking around (being routed) and arrives at the server late. Note that the 2 connections are indistinguishable (same ip-addr & port nos). The server might confuse this pkt #N from 1st stream as a segment from second stream. To avoid this confusion, we choose the SYN seq nos in such a way that there is no overlap.

But seq # space is 2^{32} bit long. If you have a large enuf file, then the seq # will be looped arnd. Pray that there is no segment still lurking arnd.

State Diagram:

..... Server

——— Client

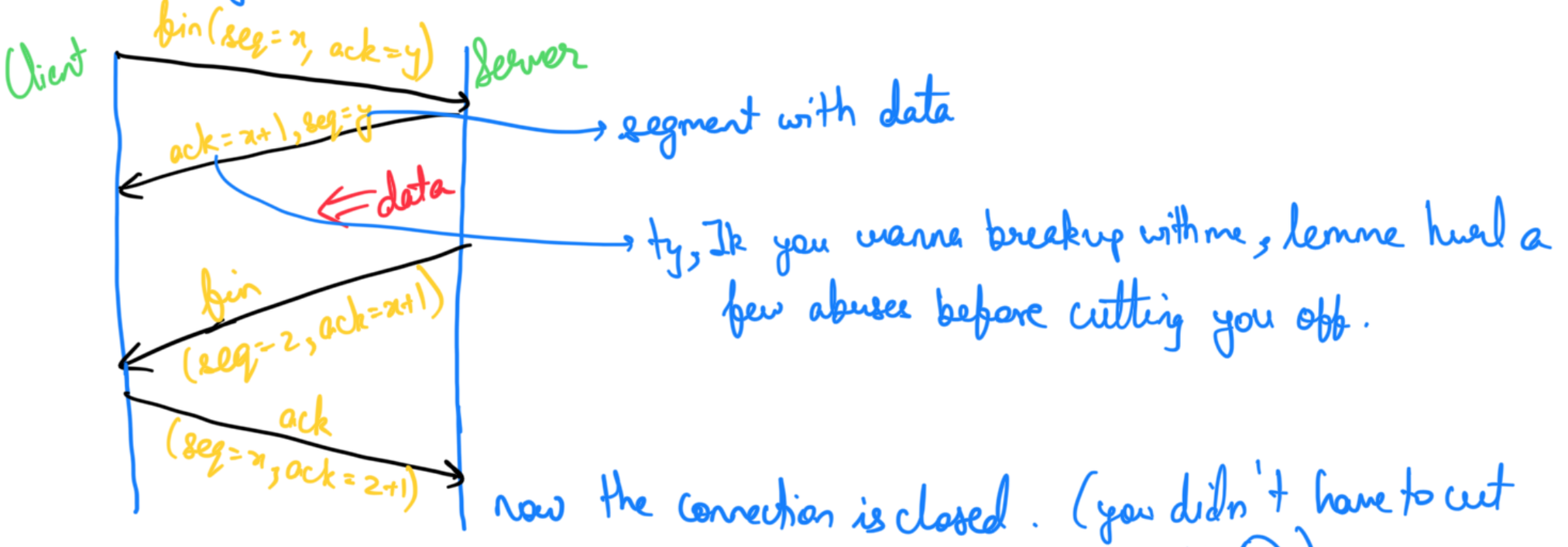


Don't be confused by the state diagram. Extraneous transitions are there to handle all other behaviours. ones in yellow.

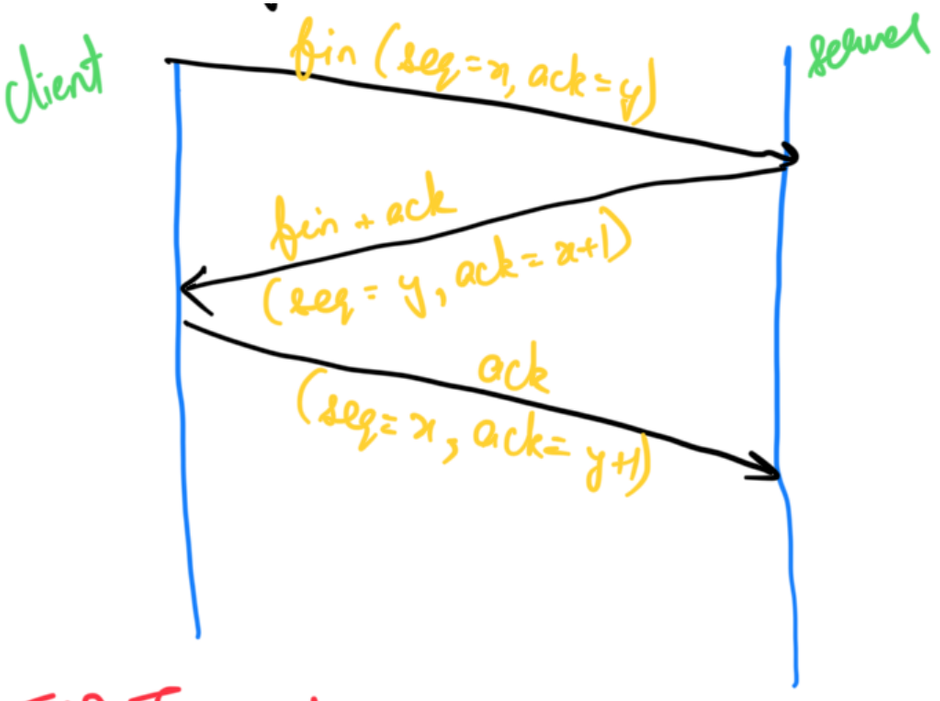
Connection Termination: (use FIN segments)

(i) Half-Close (closed by one end at a time)

When client doesn't wanna send data but server wants to; so client can't close immediately.



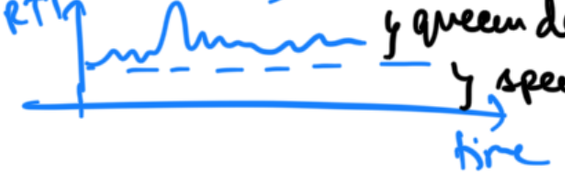
(ii) 3 way handshake termination (both close @ same time)



TCP Timeout:

Wat to do if no receive ACK? (→ retransmit) Till wen to wait for it? (→ for timeout period.)

The timeout time should be ≈> RTT. But RTT varies by orders of magnitude across network paths. Further, the RTT may not be constant cuz of queue delays and all.



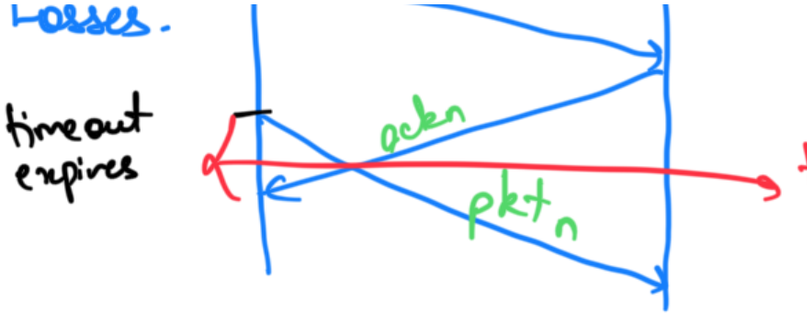
Issues:

(i) RTT varies by orders of magnitudes. (changes hugely with network paths)

(ii) Even in same path. RTT != constant. (queue delay)

(iii) 

(iii) Packet Losses.



this becomes RTT. obv nahi chalega.

OLD ALGO:

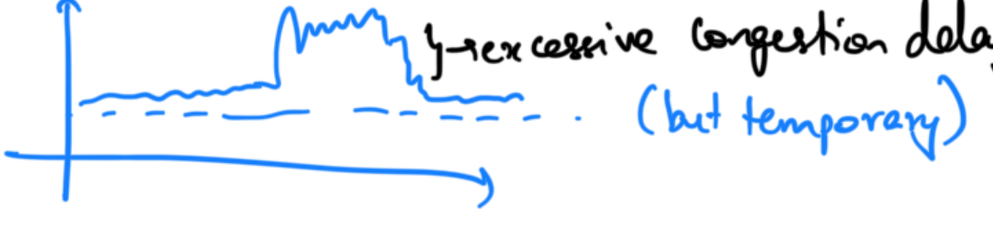
Take any RTT and set it to be timeout. How to calculate it?

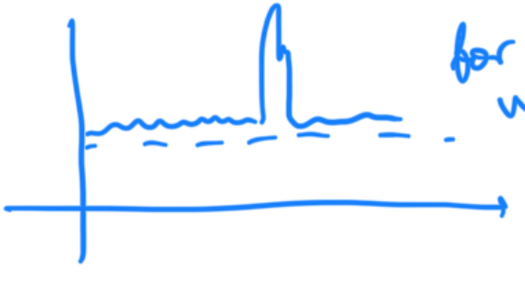
Sample RTT: most recent RTT

Estim RTT = α . Estim RTT + $(1-\alpha)$ Sample RTT. $\alpha \in (0,1)$

↳ like an avg.

if $\alpha \approx 1$; then we are ignoring sudden spikes in the RTT due to congestion.

↳ ie. if  y-excessive congestion delay. (but temporary) then inefficient.

if $\alpha \approx 0$;  for very short spike, issue as excessive waiting.

Timeout = 2x estim. RTT

This is computationally quite simple. The choosing alpha is crucial.

NEW ALGO:

Fit into $N(\mu, \sigma)$ (somewhat). Then set timeout to be $\mu + 3\sigma$ = Timeout.

$x_1, \ldots x_N$ are some samples from prob distrib of random var. X.

estim mean $= \bar{x} = \frac{1}{N} \sum_{i=1}^{N} x_i$. $\sigma = \sqrt{\frac{1}{N} \sum (x_i - \bar{x})^2}$

Mean deviation $= \frac{1}{N} \sum_{i=1}^{N} |x_i - \bar{x}|$

Diff = Estim. RTT − Sample RTT. → as before only. $\alpha \times$ estim RTT + $(1-\alpha)$ sample RTT.

Dev = Dev $(1-\beta)$ + $\beta|diff|$ → most recent dev.

↳ mean dev^

$\alpha = 7/8$; $\beta = 1/4$

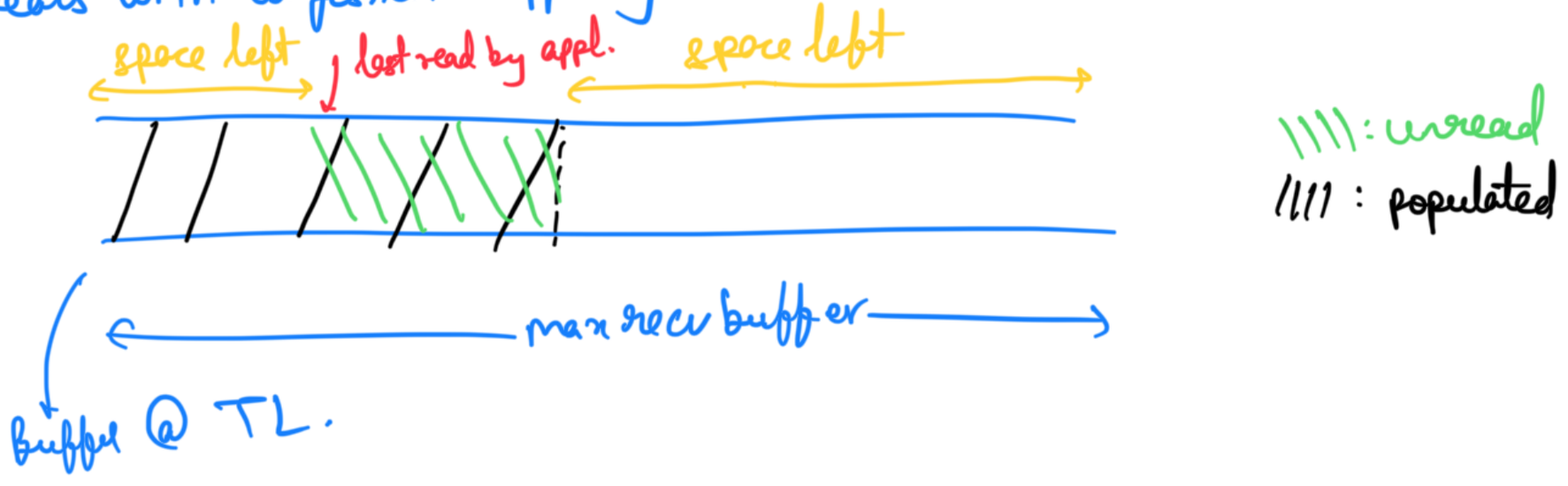
Timeout = μ (1) × estim RTT + ϕ (4) × Dev

But how to handle issue #3 ??

Sol': Do not use RTT measurement for retransmitted packet.

Flow Control:

Deals with congestion happening at receiver.



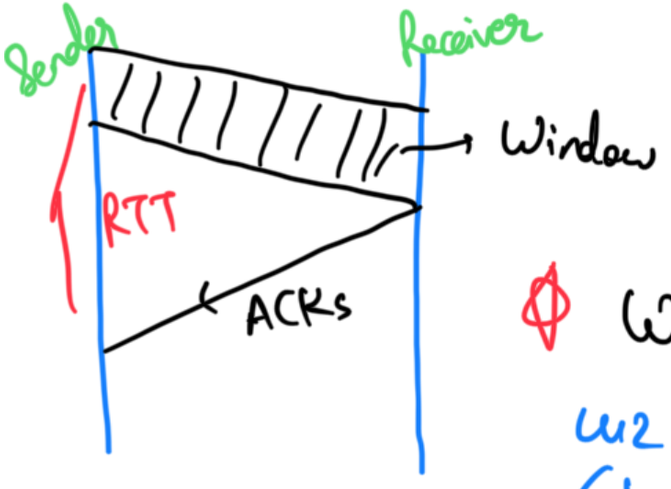
When congestion @ receiver occurs, space left is very less essentially.

The advertising window field in TCP header tells abt the space left in recv. buffer.

The sender has a "Window".

Window = max amt of data (in B) that has been sent on the network but ACK

hasn't been recd. (outstanding data)



Data Rate $\approx \frac{\text{Window}}{\text{RTT}}$

Window < Advertised Window

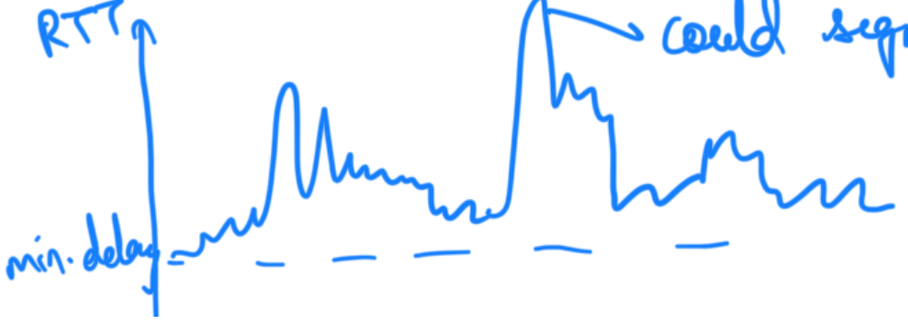
↳2 receiver doesn't have enuf space to accomodate more.
(he ain't got no blank space, baby)

Window = min (Adv. Window, Cong. Window)
- Cong. Window → congestion @ routers
- Adv. Window → will calculate in sometime

Window $\approx$ DataRate × RTT
- DataRate × RTT → delay bandwidth product.

Congestion Control: (↑ ECN (excessive congestion notif.) which we won't study on)

packets get dropped and excessive queueing delays.



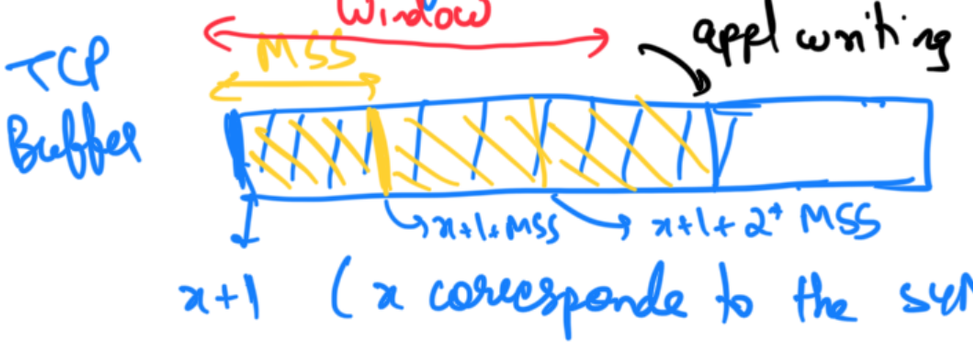
could signal congestion. But there should be a threshold for the no. of packets who have this high RTT then only can we say it is due to congestion

if there is a sudden spike, it may have occurred due to other reasons.

Further, the spike should be measured wrt min. delay & not absolute RTT cuz of varying link speeds and queue sizes.

One way to detect packet loss is via timeout.
The other way is to use ACK feedback. i.e. if ACK for a particular packet didn't reach back. (do not wait, keep sending packets, then check which packets ACK was recd)

Let us look from sender's side:



MSS: max. segment size

(x corresponds to the SYN packet)

If window = 3 x MSS; we send out 3 segments.

x+1

ACK for this segment has ACK no. x+1+MSS. (i.e. I recd all bytes till x+MSS)

We send a cumulative ack. (for each segment; a single ACK)

After the sender receives 1st ACK (for 1st segment), it shifts the window to the right. → Sliding Window Technique.

Now say that sender receives 1st, 3rd and 4th packet but the 2nd packet is dropped.

Then; ACK # in receiver buffer after receiving 1st pkt : x+1+MSS ← cuz of def^ of ACK.

3rd pkt : x+1+MSS } Duplicate ACKs.

4th pkt : x+1+MSS

Duplicate ACKs tell how many packets receiver got after first ACK but won't tell which. i.e. in this example there are 2 duplicate ACKS thus there were 2 packets successfully recd. but receiver has no way of knowing

which 2 pkts were recd. The recd packets may have been the 4th and 5th packet as well who knows.

If the receiver gets duplicate ACKs, it cannot slide the window to the right and thus re-transmits packets according to the ACKS, dup ACKs recd.

Principles of Congestion Control:

(i) If no congestion, increase congestion window (CW) conservatively.

For now assume Window = CW (ignore adv. window)

TCP uses self-clocking mechanism.

Every time an ACK arrives, we increase window by some amt.

$$\text{per ACK window increase} = \frac{\text{MSS}}{\#\text{ACKs}}$$

(#ACKs: how many ACKs you expect for the window.)

Thus, in the end; the window size increases by 1MSS per RTT.

RTT is the time between 1st packet sent out & last ACK recd in a window.

We also know that # ACKs = # segments = $\frac{CW}{MSS}$.
(per segment recd.)

$\therefore$ increase per ACK = $\frac{(MSS)^2}{CW}$.

Additive Rule ⇒ $CW \mathrel{+}= (MSS)^2 / CW$ (assuming no congestion)

(ii) If congestion is detected, decrease window size aggressively.

TCP Tahoe : $CW = 1$ on detecting congestion — TCP Vegas later
TCP Reno : $CW = CW/2$ → multiplicative decrease — TCP Africa forget

We need to decrease window size aggressively cuz there is a lot of overhead once congestion occurs. Packets are getting dropped, queuing delays increase, etc. AIMD : additive increase, multiplicative decrease.

(iii) Suppose you have a link whose speed is [illegible] but initial window

size is set to 1MSS ($\approx 10^4$ bits). Now, additive increase would result in huge amts of time being taken to transmit say a 1Gb file (could be done in 0.1s but due to slow increase; bleh).

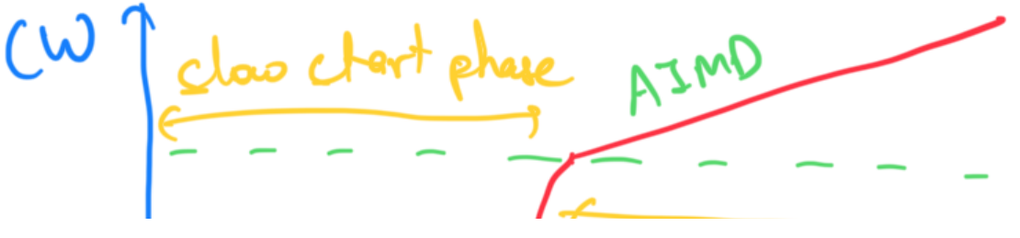
Thus, initially set CW small since we do not know appropriate data rate. Then, be aggressive to increase window initially.

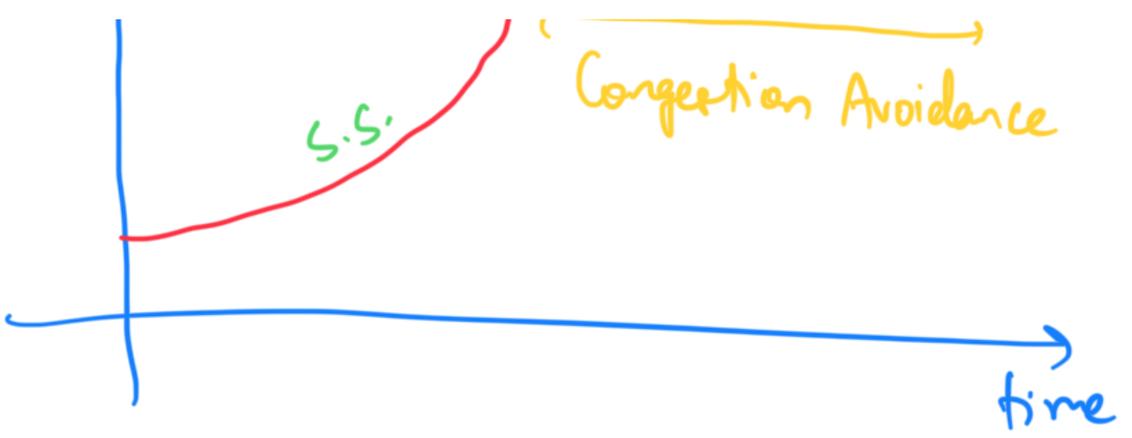
At sender: increase # ACK CW += 1MSS

Thus for $\frac{CW}{MSS}$ ACKs; CW increases by CW, thus becomes double.

This is exponential increase.

Keep in mind that sender slides window only after receiving ACK for the segment.





This is what TCP used for a long time.

RFC: request for comments.