Transmission Control Protocol

- → point to point: single sender, receiver
- → full duplex communication
- → flow control
- → connection oriented
- → cumulative ACKs
- → pipelining: congestion and flow control determine window size
- → inorder byte stream: no message boundaries

TCP segment structure

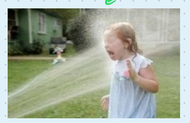
32 bits

sender port #	receiver port #
sequence number	
acknowledgement no.	
head len / unused / C E / U / A / P / R / S / F	receive window
options (variable length)	
application data (variable length)	

- byte stream no. of first byte in this segment's data
- seq # of next byte expected from other side. set(A) = this is ACK
- for flow control: no of bytes receiver willing to accept

length of TCP header
congestion notification
RST, SYN, FIN connection management

flow control



- RcvBuffer → typically 4096 bytes
- remaining buffer space updated in receiver window (rcvwnd) field
- sender limits unACKed data to rcvwnd to ensure no buffer overflow

LastByteRcvd - LastByteRead ≤ RcvBuffer

rcvwnd = RcvBuffer - [LastByteRcvd - LastByteRead]

sender, receiver

receiver

event	response
→ pkt w/ expected seq #. all data upto this already ACKed	delayed ACK. wait 500ms for next seq
→ pkt w/ expected seq # one other pending	immediate cumulative ACK
→ out of order higher than expect. gap detected	immediate send duplicate ACK
→ seg that partially/fully fills gap	immediate send ACK for this, assuming it fills from lower end

sender

create segment w/ seq # for application data, start timer if not already running for oldest unACKed segment

timeout: retransmit, restart timer

receive ACK: if ACK is for unACKed seg:
- update status
- start timer for next unACKed seg

NOTE: Fast retransmit

3 additional duplicate ACKs for a seg

⇓

retransmit without waiting for timeout

RTT, timeout interval

timeout > RTT, but how to set ?

exponential weighted moving avg (EWMA)

$$EstimatedRTT = \alpha(EstimatedRTT_{old}) + (1-\alpha)(SampleRTT)$$

where typical $\alpha = 0.125$

$$Timeout\ Interval = EstimatedRTT + 4(DevRTT)$$

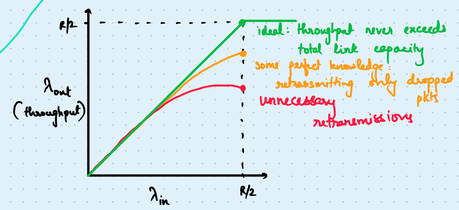
where DevRTT → DeviationRTT is given by EWMA of sample deviation from estimate

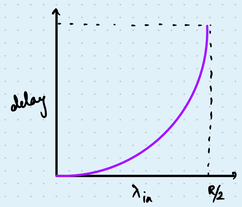
$$DevRTT = \beta(DevRTT_{old}) + (1-\beta)|EstimatedRTT - SampleRTT|$$

where typical $\beta = 0.25$

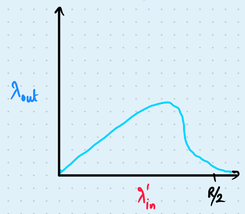
congestion control

Principles of congestion





For multiple senders (red/blue):



$\lambda_{in}' \uparrow \Rightarrow \lambda_{out} \downarrow$

$\lambda_{in}' \to R/2 \Rightarrow \lambda_{out} \to 0$

Any upstream transmission/buffering → wasted!

TCP Congestion Control

methods to deal with it

End-end

based on loss/delay, no feedback from network itself

Network assisted

routers provide direct feedback

AIMD

Additive Increase, Multiplicative Decrease

① Receive ACK ⇒ pkt received ⇓ increase speed

② Timeout ⇒ pkt lost ⇓ decrease speed

TCP rate ≈ $\frac{cwnd\ bytes/sec}{RTT}$

Slow Start

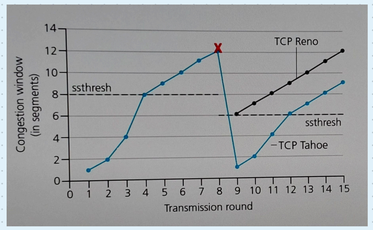
Initially cwnd = 1 MSS

Double cwnd every RTT ⇓ Done by incrementing for every ACK

Congestion Avoidance

Switch exponential increase to linear increase when cwnd gets to ½ its value before timeout

Implemented by ssthresh: set to half of cwnd on a loss event

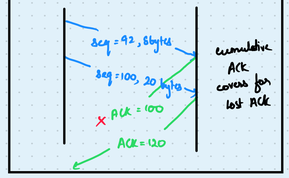


Multiplicative Decrease

3 duplicate ACKs → cut cwnd to half → TCP Reno

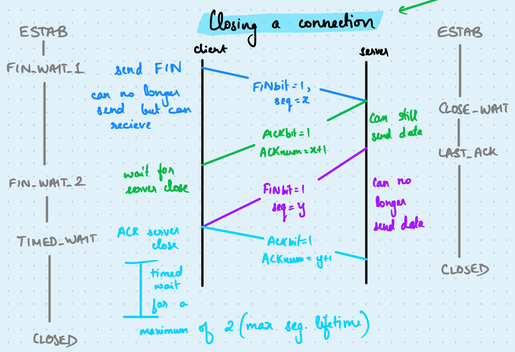
Timeout → set cwnd to only 1 MSS → TCP Tahoe

NOTE: Cumulative ACK Example



connection management

Closing a connection



Setting up a connection