Also talk about
long. ctl. for unreliable protocols.
— EBCC.

Lecture 6: 2/9/2010

Transport and Queueing

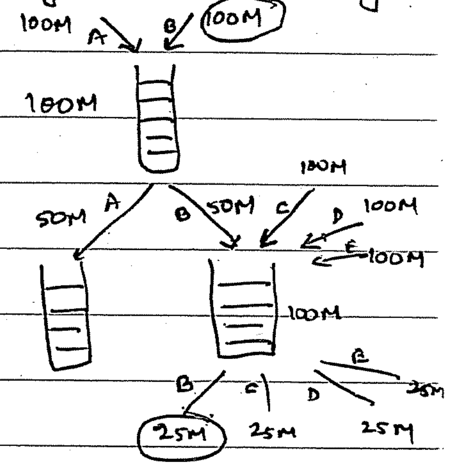
TCP: Reliable, in-order — sequence numbers
acknowledgements — for the last insequence packet received

TCP also does congestion control → regulating sending rate to match n/w capacity (or available resources)

Why can't the n/w do it?
No proper end-host support can lead to congestion collapse where the amount of useful work the n/w is doing becomes vanishingly small

↳ note: end-hosts can send as fast as their NIC allows

E.g. Fair Queueing



Flow B: 25% useful work!

Congestion control: figuring of what rate to send to get n/w capacity optimally given traffic of all other flows:

Challenges: ① Quickly figure out available head room and match it
② Back off when available capacity lowers or others want to use
③ fine time-scale adaptation.

Various ways of achieving this
TCP embodies one specific set of mechanisms.

Let's discuss the mechanisms in TCP and the reasoning behind the

(A) Slow-start → ② establish Ack clocking to help connection maintain equilibrium.

↙

① figure out available capacity (or some rough estimate of it)

MAX Total available capacity = BW x delay + Buffering = BW×D + B

① → exponential search from 0 to 2(BW×D + B)

Overshoot capacity by atmost twice.

CWND: tracks exponential growth → # packets outstanding

variable ssthresh as last estimate of capacity

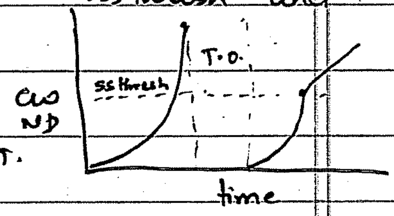
② Establish ack clocking:

at the end of search in ① above → bunch of losses → time out → lose ACK clocking

Another slowstart to quickly regain ack-clocking.

↳ upto ssthresh.

After that explore slowly between ssthresh and true available capacity. Therefore we have



CWND → guides the search

↳ amount of data outstanding every RTT.

(B) Congestion Avoidance: and control

① - How to do slow exploration?

② - How to reset estimate when congestion is experienced.

① - increase function: which next sending rate to explore.

② - decrease function: How to readjust sending rate

$R \leftarrow \alpha R + (1-\alpha) M$

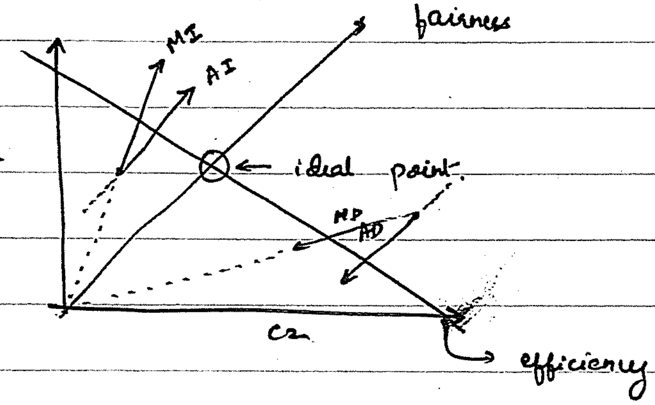
EWMA

$\beta R \rightarrow RTO.$

$RTO = R + 4 \cdot V$

choices driven by fairness ~~and~~, efficiency stability criteria.

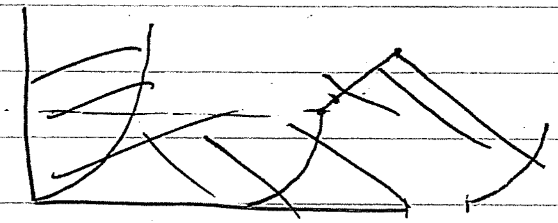
Two sender

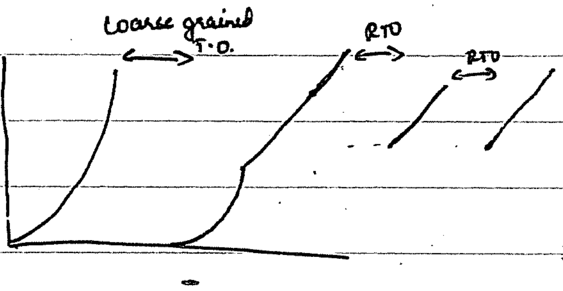


AI – MD converges to fairness and efficiency.

AI = 1 MD = 1/2.

So the overall algorithm.





③ Feedback from timeouts, losses and Acks.

↙

duplicate acknowledgements to figure out loss → fast retransmit

Waiting for time-out and half cwnd full of packets to empty → loss of Ack clocking

Using incoming Acks to clock out packets

1 – for every other Ack

(RENO).