lecture #2 TCP

- Packet dropped if queue full → delay finite
- Packet typically needs to be retransmitted → extra work for endhosts and routers
- Congestion control in Networks - summary
  - Sending pkts into the network at too high rate causes congestion.
  - Consequences:
    - Long queues in routers → long delays
    - Overflow of queues in routers → pkt loss
    - Lost pkt needs to be retransmitted
    - If pkt is dropped at some later router, the bandwidth spent on getting it that far is wasted.
  - Long delay may cause timeout, retransmission then is unnecessary duplicate.

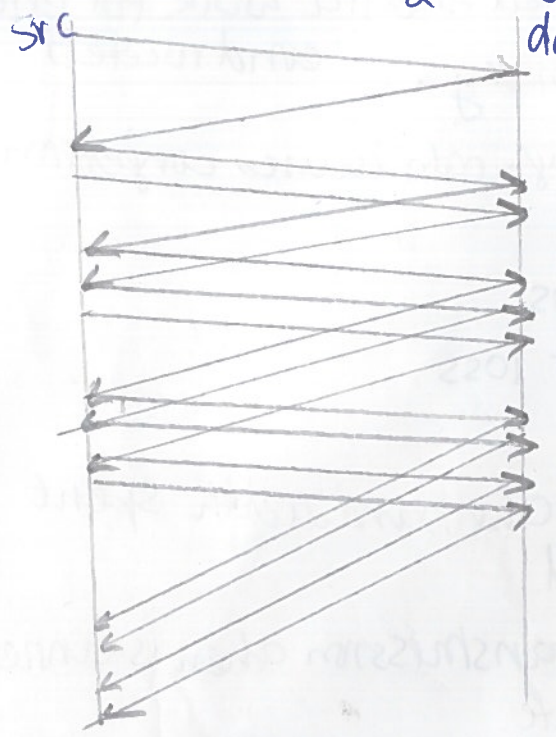
→ infinite buffers are not solution

Without feedback, network may suffer from congestion collapse

- Classification of congestion control approaches
  - router-centric vs host-centric
  - reservation-based vs feedback-based
  - window-based vs rate-based
  - We need algorithm that:
    - can learn the optimal data rate to send at
    - can adapt to changes
    - shares bandwidth in a fair way
- Congestion control in TCP
  - host-centric, feedback-based, window-based
  - pkt drop is treated as sign of congestion
  - hosts adjust their congestion window based on observations
  - hosts send no more data than allowed by congestion and receiver win
  - congestion window is an upper bound on the amount of data a TCP connection has outstanding in the network, and thus on the amount of the router buffer space it occupies

• Additive increase, multiplicative decrease (AIMD)

• If no loss during RTT, increase Congestion Window by 1 MSS.
If loss occurs → $\frac{1}{2}$ Cong. win



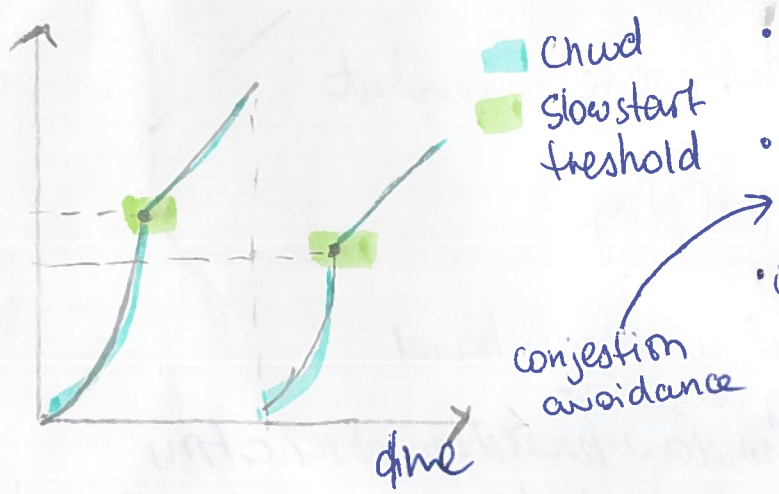
• TCP maintains cwnd in bytes
• MSS – max. segment size
• additive increase:
increase cwnd by 1 MSS every RTT
• ≈ by $MSS \times \frac{MSS}{cwnd}$ for each Acked MSS

• rate = $\frac{\sqrt{3/2} \cdot MSS}{RTT \cdot \sqrt{p}}$

• A non-TCP flow is "TCP friendly" if it does not use more bandwidth than TCP flow would do in the same circumstances

• Slow start

• to find available capacity of network
•

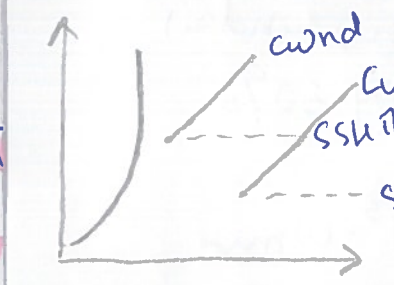


• if below ssthr and no loss:
cwnd = 2 · cwnd every RTT
• if above ssthr and no loss:
cwnd += 1 MSS every RTT
• if loss ss timeout:
ssthr = $\frac{cwnd}{2}$, cwnd = 1

• TCP maintains cwnd in byte
• slow start:
double cwnd every RTT ≈ increase cwnd by 1 MSS for each Acked MSS

Fast retransmit, Fast recovery

• to keep data flow after single pkt loss
• Fast retransmit: after 3 duplicate Acks → retransmit pkt
Fast recovery: skip slow start after fast retransmit



~~ssthr = cwnd/2~~
~~after 3 dup Acks~~

after 3 dup Acks
ssthr = $\frac{cwnd}{2}$
cwnd = $\frac{cwnd}{2}$ + 3 MSS

for every further dup Ack →
→ cwnd += 1 MSS

first non-dup Ack: (Ack for retransmitted pkt)
cwnd = ssthr and resume normal operation

Summary of Congestion Control TCP (new Reno)

slow start
• while cwnd < ssthr, cwnd += 1 MSS for each Acked MSS

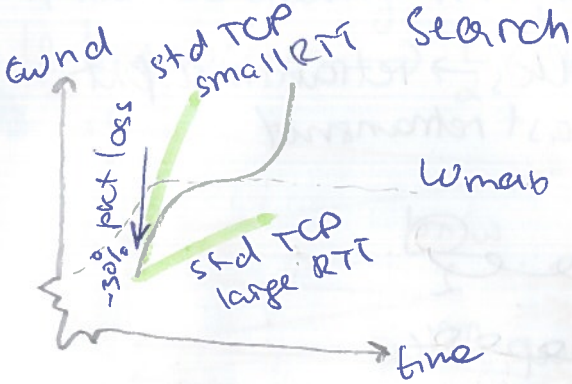
Congestion avoidance (AIMD)
• if cwnd ≥ sshr; cwnd += MSS × $\frac{MSS}{cwnd}$ for each Acked MSS

Fast recovery
• After 3 dup Acks; ssthr = $\frac{cwnd}{2}$, cwnd = $\frac{cwnd}{2}$ + 3 MSS
• For every further dup Ack; cwnd += 1 MSS
• First non-dup ACK, cwnd = ssthr and resume normal operation

Time-out
• after timeout, ssthr = $\frac{cwnd}{2}$, cwnd = 1 MSS and resume slow start

TCP CUBIC

- treat the bandwidth estimation as binary (not linear) search problem



- upon 3 dup Acks:
  - ~~set~~ W_{max} = previous window
  - reduce window by 30%
- Otherwise:

$$W(t) = 0.4 \cdot (t-K)^3 + W_{max}$$

t - time since last 3 dup Acks

$K \rightarrow W(0) = W_{max} - 30\%$

- if $w(t) <$ window of standard TCP $\rightarrow$ do standard TCP