15-441 Computer Networking

Lecture 9 – More TCP & Congestion Control

Overview

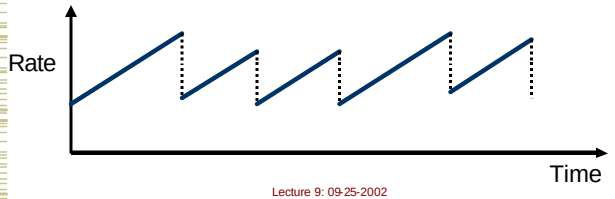
- TCP congestion control
- TCP modern loss recovery
- TCP modeling

TCP Congestion Control

- Changes to TCP motivated by ARPANET congestion collapse
- Basic principles
  - AIMD
  - Packet conservation
  - Reaching steady state quickly
  - ACK clocking

AIMD

- Distributed, fair and efficient
- Packet loss is seen as sign of congestion and results in a multiplicative rate decrease
  - Factor of 2
- TCP periodically probes for available bandwidth by increasing its rate



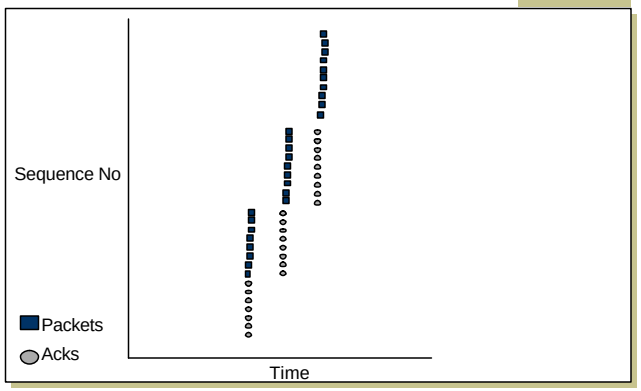
Implementation Issue

- Operating system timers are very coarse – how to pace packets out smoothly?
- Implemented using a congestion window that limits how much data can be in the network.
  - TCP also keeps track of how much data is in transit
- Data can only be sent when the amount of outstanding data is less than the congestion window.
  - The amount of outstanding data is increased on a “send” and decreased on “ack”
  - (last sent – last acked) < congestion window
- Window limited by both congestion and buffering
  - Sender’s maximum window = Min (advertised window, cwnd)

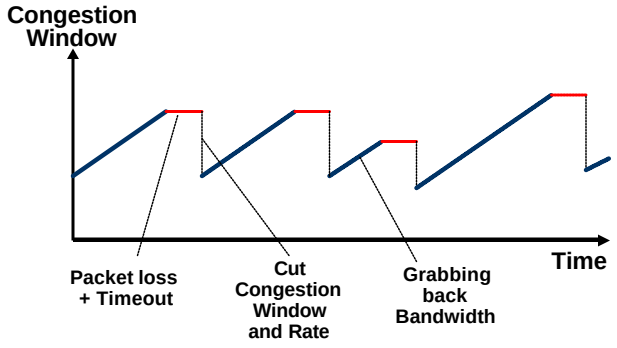
Congestion Avoidance

- If loss occurs when cwnd = W
  - Network can handle 0.5W ~ W segments
  - Set cwnd to 0.5W (multiplicative decrease)
- Upon receiving ACK
  - Increase cwnd by 1/cwnd
- Implements AIMD

Congestion Avoidance Sequence Plot



Congestion Avoidance Behavior

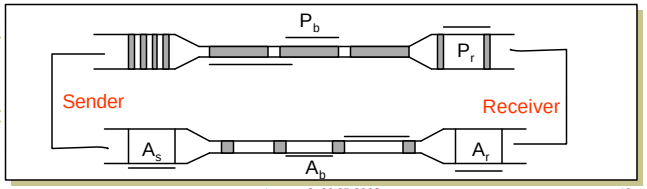


Packet Conservation

- Packet conservation == at equilibrium, inject packet into network only when one is removed
  - Sliding window and not rate controlled
  - But still need to avoid sending burst of packets → would overflow links
    - Need to carefully pace out packets
- Helps provide stability
- Need to eliminate spurious retransmissions
  - Accurate RTO estimation
  - Better loss recovery techniques (e.g. fast retransmit)

TCP Packet Pacing

- Congestion window helps to "pace" the transmission of data packets
- In steady state, a packet is sent when an ack is received
  - Data transmission remains smooth, once it is smooth
  - Self-clocking behavior

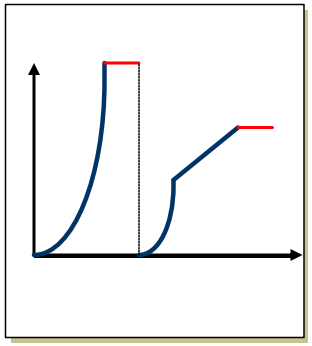


Reaching Steady State

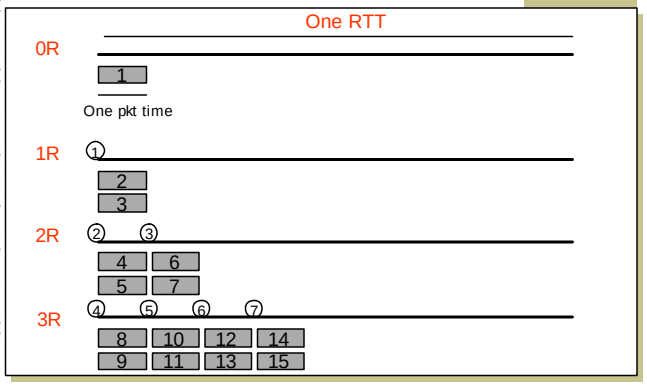
- Doing AIMD is fine in steady state but slow…
- How does TCP know what is a good initial rate to start with?
  - Should work both for a CDPD (10s of Kbps or less) and for supercomputer links (2.4 Gbps and growing)
- Quick initial phase to help get up to speed (slow start)

Slow Start Packet Pacing

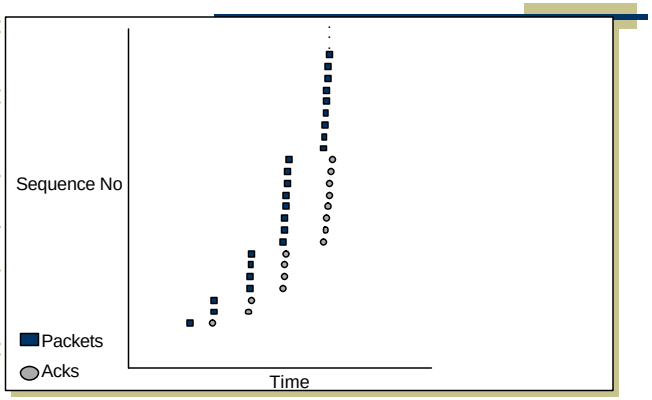
- How do we get this clocking behavior to start?
  - Initialize cwnd = 1
  - Upon receipt of every ack, cwnd = cwnd + 1
- Implications
  - Window actually increases to W in RTT * $\log_2(W)$
  - Can overshoot window and cause packet loss



Slow Start Example



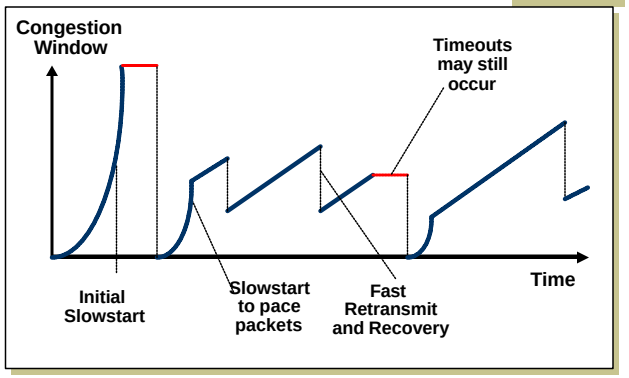
Slow Start Sequence Plot



Return to Slow Start

- If packet is lost we lose our self clocking as well
  - Need to implement slow-start and congestion avoidance together
- When timeout occurs set ssthresh to 0.5w
  - If cwnd < ssthresh, use slow start
  - Else use congestion avoidance

TCP Saw Tooth Behavior



Overview

- TCP congestion control
- TCP modern loss recovery
- TCP modeling

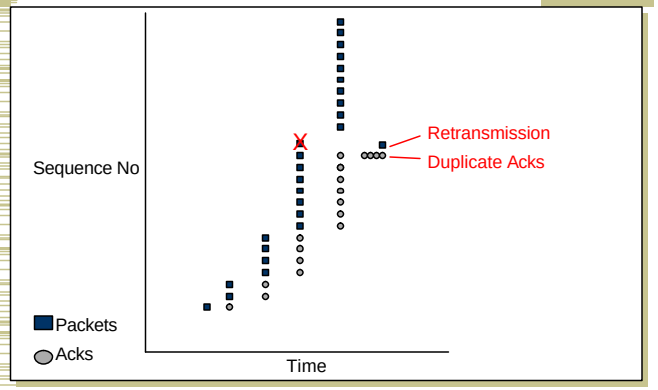
TCP Flavors

- Tahoe, Reno, Vegas
- TCP Tahoe (distributed with 4.3BSD Unix)
  - Original implementation of Van Jacobson's mechanisms (VJ paper)
  - Includes:
    - Slow start
    - Congestion avoidance
    - Fast retransmit

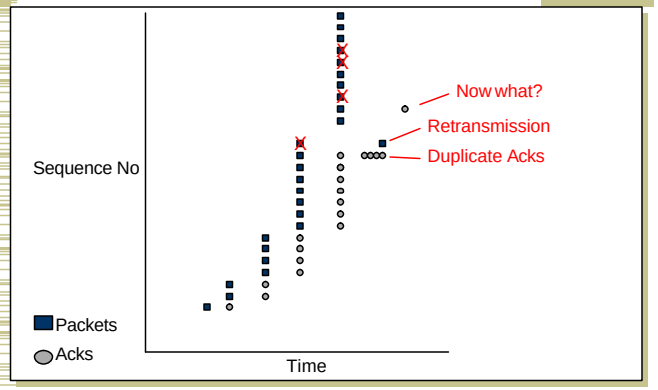
Fast Retransmit

- What are duplicate acks (dupacks)?
  - Repeated acks for the same sequence
- When can duplicate acks occur?
  - Loss
  - Packet re-ordering
  - Window update – advertisement of new flow control window
- Assume re-ordering is infrequent and not of large magnitude
  - Use receipt of 3 or more duplicate acks as indication of loss
  - Don't wait for timeout to retransmit packet

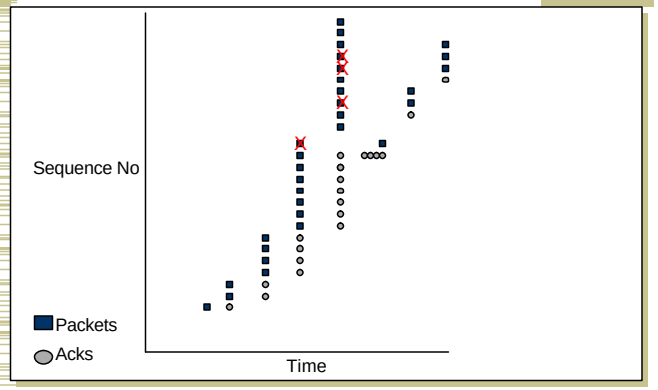
Fast Retransmit



Multiple Losses



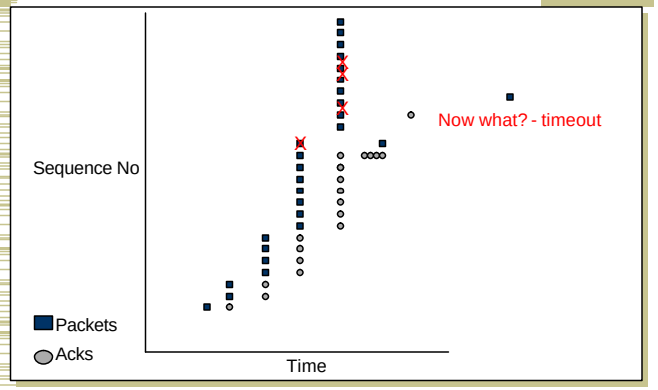
Tahoe



TCP Reno (1990)

- All mechanisms in Tahoe
- Addition of fast-recovery
  - Opening up congestion window after fast retransmit
- Delayed acks
- Header prediction
  - Implementation designed to improve performance
  - Has common case code inlined
- With multiple losses, Reno typically timeouts because it does not see duplicate acknowledgements

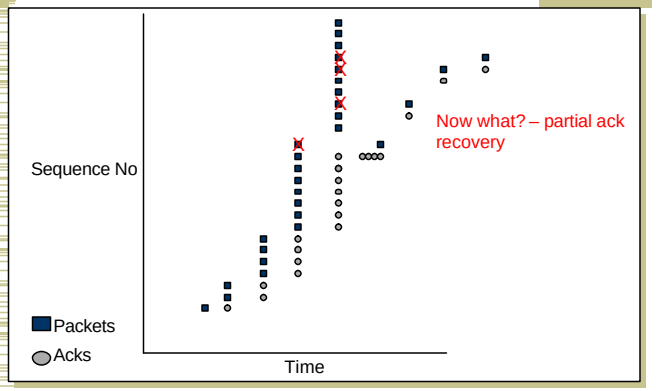
Reno



NewReno

- The ack that arrives after retransmission (partial ack) could indicate that a second loss occurred
- When does NewReno timeout?
  - When there are fewer than three dupacks for first loss
  - When partial ack is lost
- How fast does it recover losses?
  - One per RTT

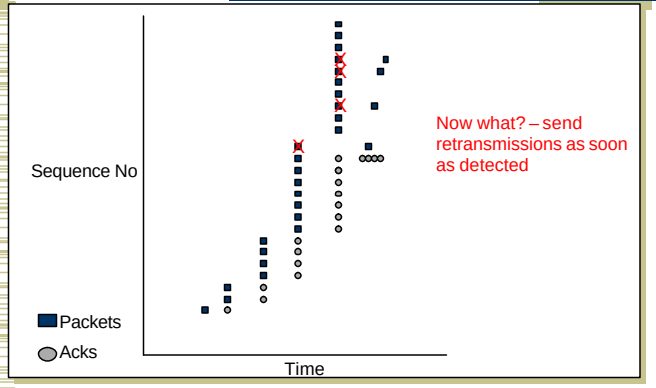
NewReno



SACK

- Basic problem is that cumulative acks provide little information
  - Ack for just the packet received
    - What if acks are lost? → carry cumulative also
    - Not used
  - Bitmask of packets received
    - Selective acknowledgement (SACK)
- How to deal with reordering

SACK



Performance Issues

- Timeout >> fast rexmit
  - Need 3 dupacks/sacks
  - Not great for small transfers
    - Don't have 3 packets outstanding
  - What are real loss patterns like?
- How to deal with reordering?

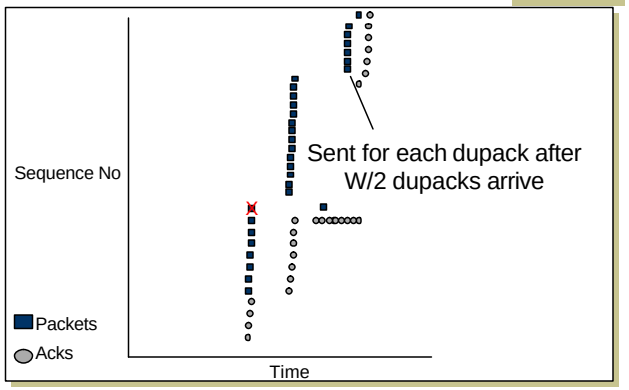
How to Change Window

- When a loss occurs have W packets outstanding
- New cwnd = 0.5 * cwnd
  - How to get to new state?

Fast Recovery

- Each duplicate ack notifies sender that single packet has cleared network
- When < cwnd packets are outstanding
  - Allow new packets out with each new duplicate acknowledgement
- Behavior
  - Sender is idle for some time – waiting for ½ cwnd worth of dupacks
  - Transmits at original rate after wait
    - Ack clocking rate is same as before loss

Fast Recovery



Overview

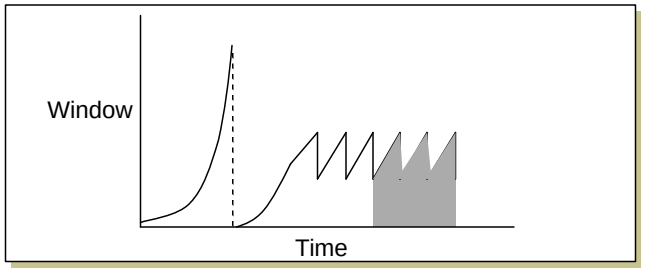
- TCP congestion control
- TCP modern loss recovery
- TCP modeling

TCP Modeling

- Given the congestion behavior of TCP can we predict what type of performance we should get?
- What are the important factors
  - Loss rate
    - Affects how often window is reduced
  - RTT
    - Affects increase rate and relates BW to window
  - RTO
    - Affects performance during loss recovery
  - MSS
    - Affects increase rate

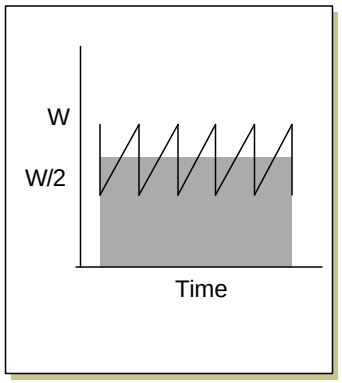
Overall TCP Behavior

- Let's concentrate on steady state behavior with no timeouts and perfect loss recovery
- Packets transferred = area under curve



Transmission Rate

- What is area under curve?
  - A = avg window * time = ¾ W * T
- What was bandwidth?
  - BW = A / T = ¾ W
    - In packets per RTT
  - Need to convert to bytes per second
  - BW = ¾ W * MSS / RTT
- What is W?
  - Depends on loss rate



Simple TCP Model

- Some additional assumptions
  - Fixed RTT
  - No delayed ACKs
- In steady state, TCP losses packet each time window reaches W packets
  - Window drops to W/2 packets
  - Each RTT window increases by 1 packet→W/2 * RTT before next loss

Simple Loss Model

- What was the loss rate?
  - Packets transferred = (¾ W/RTT) * (W/2 * RTT) = $3W^2/8$
  - 1 packet lost → loss rate = p = $8/3W^2$
  - $W = \sqrt{\frac{8}{3p}}$
- BW = ¾ * W * MSS / RTT
  - $BW = \frac{MSS}{RTT \times \sqrt{2p/3}}$

TCP Friendliness

- What does it mean to be TCP friendly?
  - TCP is not going away
  - Any new congestion control must compete with TCP flows
    - Should not clobber TCP flows and grab bulk of link
    - Should also be able to hold its own, i.e. grab its fair share, or it will never become popular
- How is this quantified/shown?
  - Has evolved into evaluating loss/throughput behavior
  - If it shows 1/sqrt(p) behavior it is ok
  - But is this really true?

Next Lecture

- Workload changes
- TCP & routers
- TCP options