15-441 Computer Networking

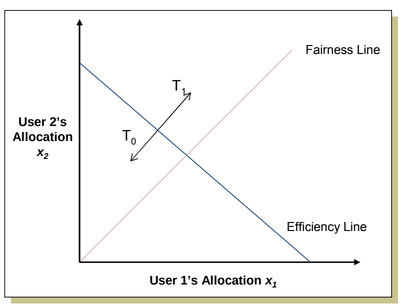
Lecture 20 – TCP Performance

Outline

- TCP congestion avoidance
- TCP slow start
- TCP modeling

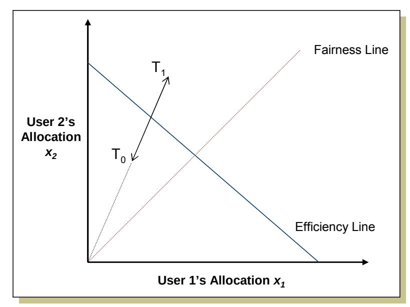
Additive Increase/Decrease

- Both X_1 and X_2 increase/ decrease by the same amount over time
  - Additive increase improves fairness and additive decrease reduces fairness



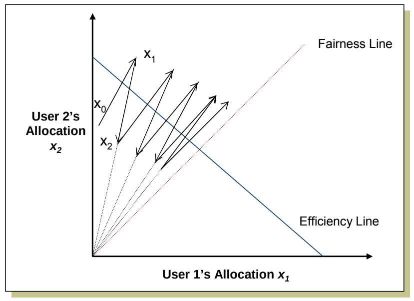
Muliplicative Increase/Decrease

- Both X_1 and X_2 increase by the same factor over time
  - Extension from origin – constant fairness



What is the Right Choice?

- Constraints limit us to AIMD
  - Improves or keeps fairness constant at each step
  - AIMD moves towards optimal point

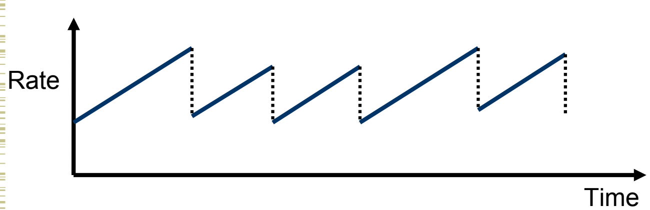


TCP Congestion Control

- 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



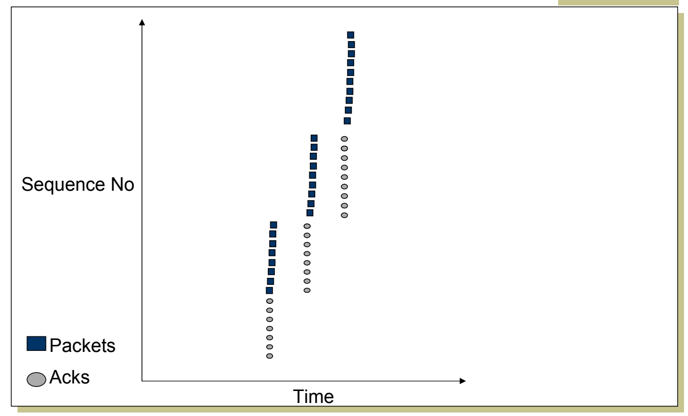
Implementation Issue

- 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)

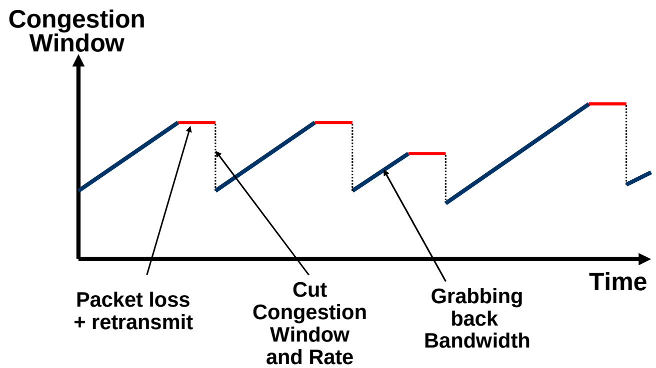
Congestion Avoidance

- 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 packet)/cwnd
    - What is 1 packet? → 1 MSS worth of bytes
    - After cwnd packets have passed by → approximately increase of 1 MSS
- Implements AIMD

Congestion Avoidance Sequence Plot



Congestion Avoidance Behavior

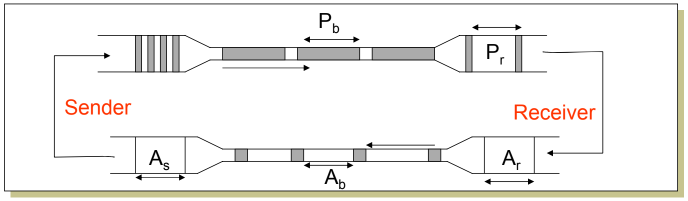


Packet Conservation

- 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



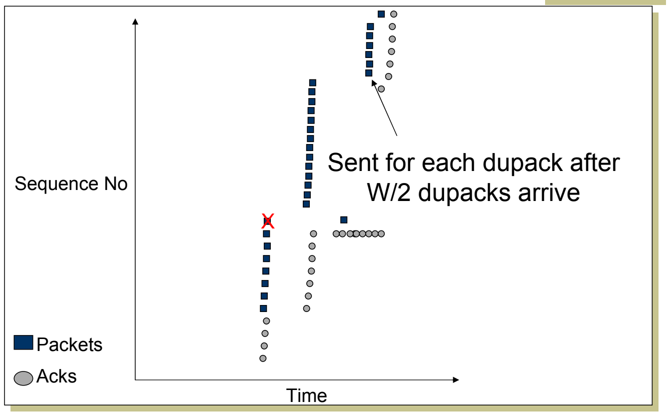
How to Change Window

- When a loss occurs have W packets outstanding
- New cwnd = 0.5 * cwnd
  - How to get to new state without losing ack clocking?

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

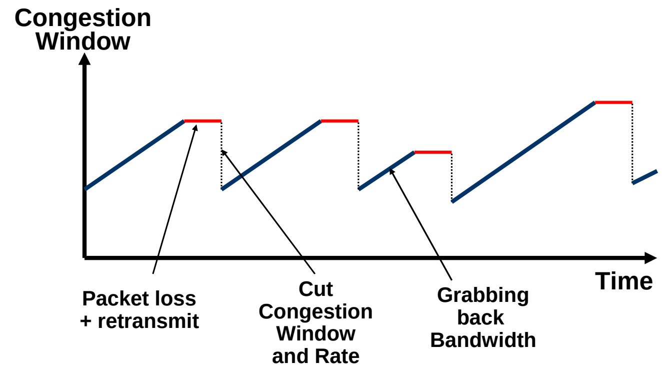
Fast Recovery



Outline

- TCP congestion avoidance
- TCP slow start
- TCP modeling

Congestion Avoidance Behavior

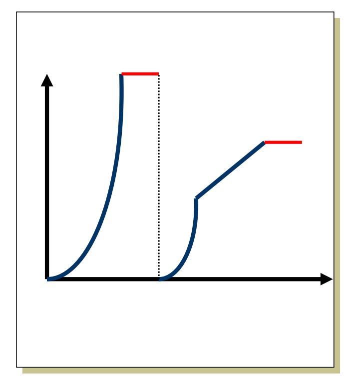


Reaching Steady State

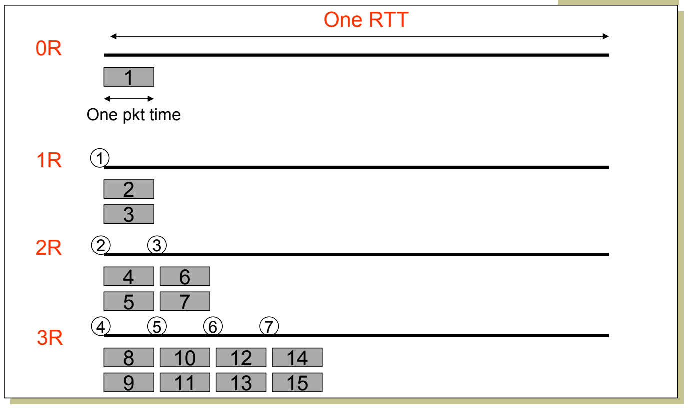
- Doing AIMD is fine in steady state but slow…
- How does TCP know what is a good initial rate to start with?
  - Should work both for a CDPD (10s of Kbps or less) and for supercomputer links (10 Gbps and growing)
- Quick initial phase to help get up to speed (slow start)

Slow Start Packet Pacing

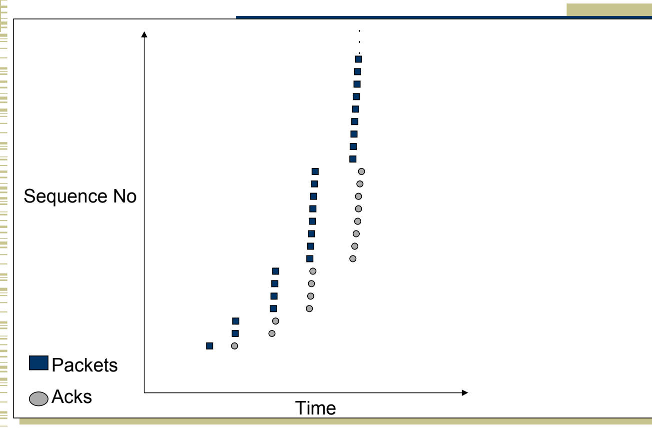
- How do we get this clocking behavior to start?
  - Initialize cwnd = 1
  - Upon receipt of every ack, cwnd = cwnd + 1
- Implications
  - Window actually increases to W in RTT * $\log_2(W)$
  - Can overshoot window and cause packet loss



Slow Start Example



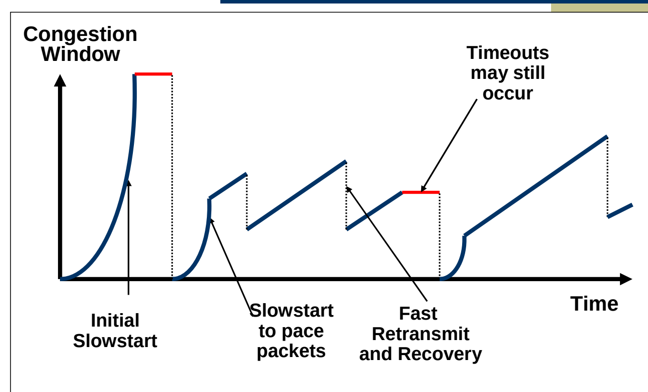
Slow Start Sequence Plot



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 retransmission occurs set ssthresh to 0.5w
  - If cwnd < ssthresh, use slow start
  - Else use congestion avoidance

TCP Saw Tooth Behavior



Outline

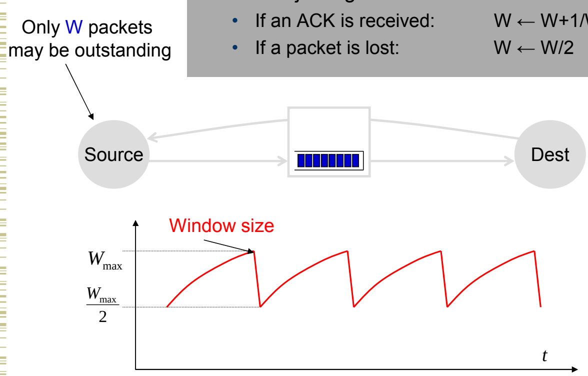
- TCP congestion avoidance
- TCP slow start
- TCP modeling

TCP Performance

- Can TCP saturate a link?
- Congestion control
  - Increase utilization until… link becomes congested
  - React by decreasing window by 50%
  - Window is proportional to rate * RTT
- Doesn't this mean that the network oscillates between 50 and 100% utilization?
  - Average utilization = 75%??
  - No…this is *not* right!

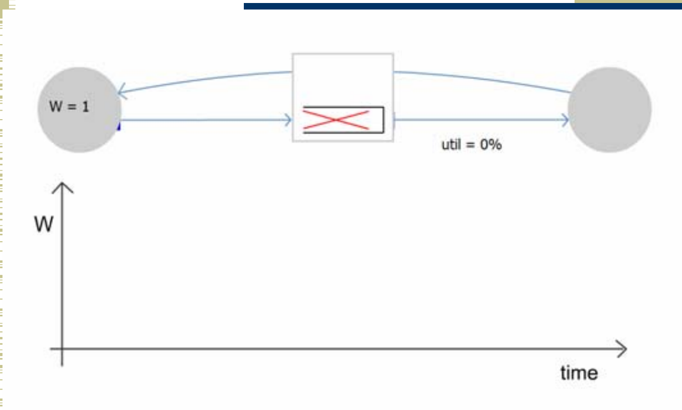
TCP Congestion Control

Rule for adjusting W
- If an ACK is received: W ← W+1/W
- If a packet is lost: W ← W/2

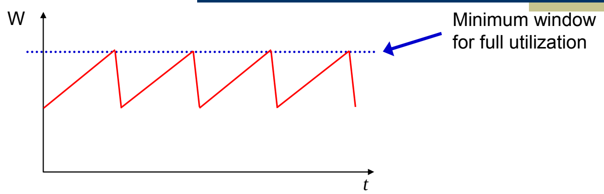


Single TCP Flow

Router without buffers



Summary Unbuffered Link



- The router can't fully utilize the link
  - If the window is too small, link is not full
  - If the link is full, next window increase causes drop
  - With no buffer it still achieves 75% utilization

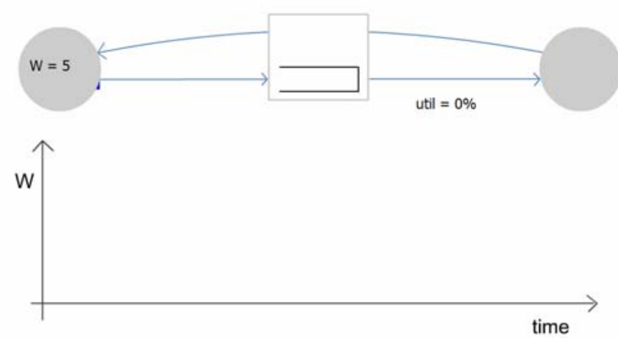
TCP Performance

- In the real world, router queues play important role
  - Window is proportional to rate * RTT
    - But, RTT changes as well the window
  - Window to fill links = propagation RTT * bottleneck bandwidth
    - If window is larger, packets sit in queue on bottleneck link

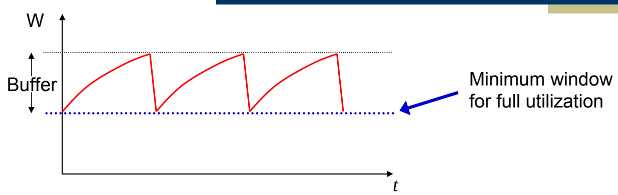
TCP Performance

- If we have a large router queue → can get 100% utilization
  - But, router queues can cause large delays
- How big does the queue need to be?
  - Windows vary from W → W/2
    - Must make sure that link is always full
    - W/2 > RTT * BW
    - W = RTT * BW + Qsize
    - Therefore, Qsize > RTT * BW
  - Ensures 100% utilization
  - Delay?
    - Varies between RTT and 2 * RTT

Single TCP Flow
Router with large enough buffers for full link utilization



Summary Buffered Link



- With sufficient buffering we achieve full link utilization
  - The window is always above the critical threshold
  - Buffer absorbs changes in window size
    - Buffer Size = Height of TCP Sawtooth
    - Minimum buffer size needed is 2T*C
  - This is the origin of the rule-of-thumb

TCP (Summary)

- General loss recovery
  - Stop and wait
  - Selective repeat
- TCP sliding window flow control
- TCP state machine
- TCP loss recovery
  - Timeout-based
    - RTT estimation
  - Fast retransmit
  - Selective acknowledgements

TCP (Summary)

- Congestion collapse
  - Definition & causes
- Congestion control
  - Why AIMD?
  - Slow start & congestion avoidance modes
  - ACK clocking
  - Packet conservation
- TCP performance modeling
  - How does TCP fully utilize a link?
    - Role of router buffers

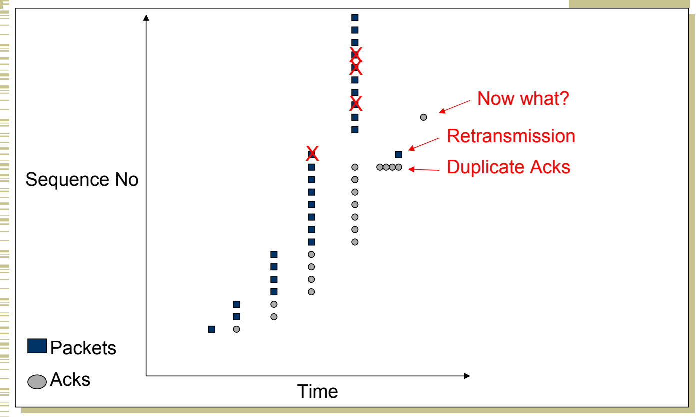
EXTRA SLIDES

The rest of the slides are FYI

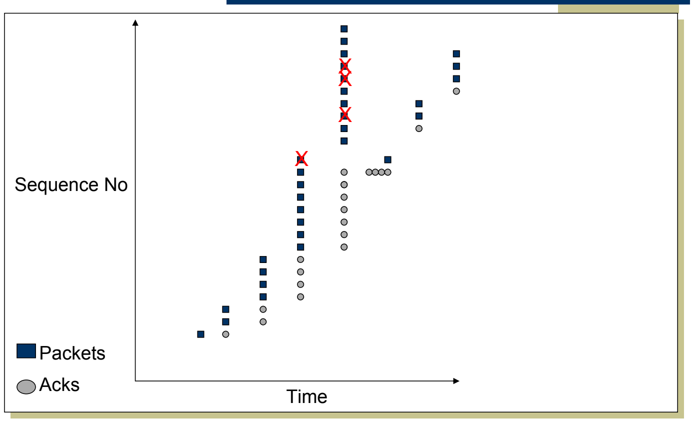
TCP Variations

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

Multiple Losses



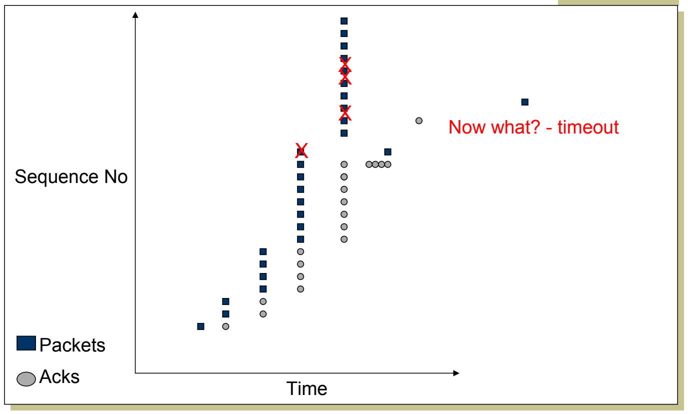
Tahoe



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

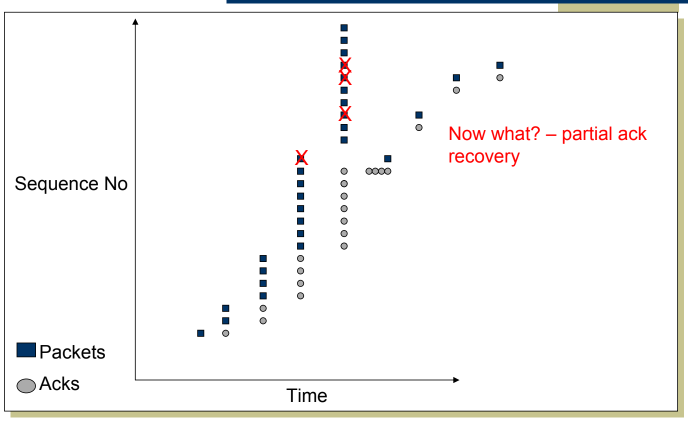
Reno



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



Changing Workloads

- New applications are changing the way TCP is used
- 1980's Internet
  - Telnet & FTP → long lived flows
  - Well behaved end hosts
  - Homogenous end host capabilities
  - Simple symmetric routing
- 2000's Internet
  - Web & more Web → large number of short xfers
  - Wild west – everyone is playing games to get bandwidth
  - Cell phones and toasters on the Internet
  - Policy routing

Short Transfers

- Fast retransmission needs at least a window of 4 packets
  - To detect reordering
- Short transfer performance is limited by slow start → RTT

Short Transfers

- Start with a larger initial window
- What is a safe value?
  - TCP already burst 3 packets into network during slow start
  - Large initial window = min (4*MSS, max (2*MSS, 4380 bytes)) [rfc2414]
    - Not a standard yet
  - Enables fast retransmission
  - Only used in initial slow start not in any subsequent slow start

Well Behaved vs. Wild West

- How to ensure hosts/applications do proper congestion control?
- Who can we trust?
  - Only routers that we control
  - Can we ask routers to keep track of each flow
    - Per flow information at routers tends to be expensive
    - Fair-queuing later in the semester

TCP Fairness Issues

- Multiple TCP flows sharing the same bottleneck link do not necessarily get the same bandwidth.
  - Factors such as roundtrip time, small differences in timeouts, and start time, … affect how bandwidth is shared
  - The bandwidth ratio typically does stabilize
- Users can grab more bandwidth by using parallel flows.
  - Each flow gets a share of the bandwidth to the user gets more bandwidth than users who use only a single flow

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?

Overview

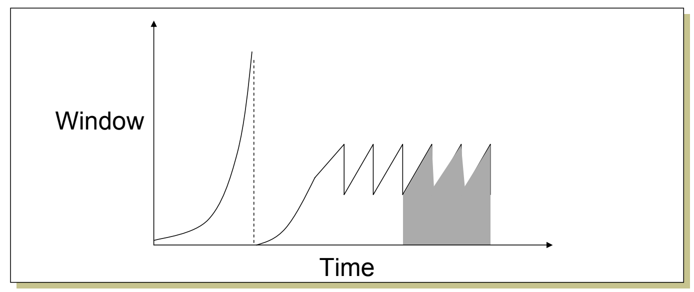
- TCP variants
- TCP modeling
- TCP details

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

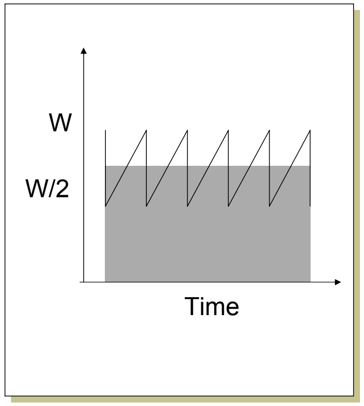
Overall TCP Behavior

- 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?
  - W = pkts/RTT, T = RTTs
  - 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
  - $W = \sqrt{\frac{8}{3p}} = \frac{4}{3} \times \sqrt{\frac{3}{2p}}$
  - $BW = \frac{MSS}{RTT \times \sqrt{2p/3}}$

Fairness

- BW proportional to 1/RTT?
- Do flows sharing a bottleneck get the same bandwidth?
  - NO!
- TCP is RTT fair
  - If flows share a bottleneck and have the same RTTs then they get same bandwidth
  - Otherwise, in inverse proportion to the RTT

Overview

- TCP variants
- TCP modeling
- TCP details

Delayed ACKS

- Problem:
  - In request/response programs, you send separate ACK and Data packets for each transaction
- Solution:
  - Don't ACK data immediately
  - Wait 200ms (must be less than 500ms – why?)
  - Must ACK every other packet
  - Must not delay duplicate ACKs

TCP ACK Generation [RFC 1122, RFC 2581]

Event	TCP Receiver action
In-order segment arrival, No gaps, Everything else already ACKed	Delayed ACK. Wait up to 500ms for next segment. If no next segment, send ACK
In-order segment arrival, No gaps, One delayed ACK pending	Immediately send single cumulative ACK
Out-of-order segment arrival Higher-than-expect seq. # Gap detected	Send duplicate ACK, indicating seq. # of next expected byte
Arrival of segment that partially or completely fills gap	Immediate ACK

Delayed Ack Impact

- TCP congestion control triggered by acks
  - If receive half as many acks → window grows half as fast
- Slow start with window = 1
  - Will trigger delayed ack timer
  - First exchange will take at least 200ms
  - Start with > 1 initial window
    - Bug in BSD, now a "feature"/standard

Nagel's Algorithm

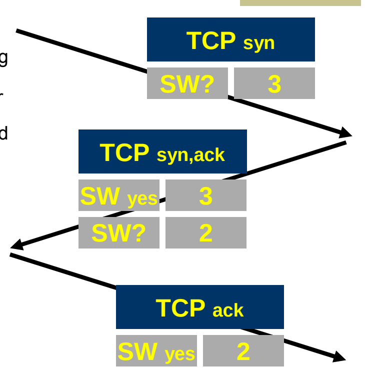
- Small packet problem:
  - Don't want to send a 41 byte packet for each keystroke
  - How long to wait for more data?
- Solution:
  - Allow only one outstanding small (not full sized) segment that has not yet been acknowledged
  - Can be disabled for interactive applications

Large Windows

- Delay-bandwidth product for 100ms delay
  - 1.5Mbps: 18KB
  - 10Mbps: 122KB
  - 45Mbps: 549KB
  - 100Mbps: 1.2MB
  - 622Mbps: 7.4MB
  - 1.2Gbps: 14.8MB
- Why is this a problem?
  - 10Mbps > max 16bit window
- Scaling factor on advertised window
  - Specifies how many bits window must be shifted to the left
  - Scaling factor exchanged during connection setup

Window Scaling: Example Use of Options

- "Large window" option (RFC 1323)
  - Negotiated by the hosts during connection establishment
  - Option 3 specifies the number of bits by which to shift the value in the 16 bit window field
  - Independently set for the two transmit directions
- The scaling factor specifies bit shift of the window field in the TCP header
  - Scaling value of 2 translates into a factor of 4
- Old TCP implementations will simply ignore the option
  - Definition of an option



Maximum Segment Size (MSS)

- Problem: what packet size should a connection use?
- Exchanged at connection setup
  - Uses a TCP option
  - Typically pick MTU of local link
- What all does this effect?
  - Efficiency
  - Congestion control
  - Retransmission
- Path MTU discovery
  - Why should MTU match MSS?

Silly Window Syndrome

- Problem: (Clark, 1982)
  - If receiver advertises small increases in the receive window then the sender may waste time sending lots of small packets
- Solution
  - Receiver must not advertise small window increases
  - Increase window by min(MSS,RecvBuffer/2)

Protection From Wraparound

- Wraparound time vs. Link speed
  - 1.5Mbps: 6.4 hours
  - 10Mbps: 57 minutes
  - 45Mbps: 13 minutes
  - 100Mbps: 6 minutes
  - 622Mbps: 55 seconds
  - 1.2Gbps: 28 seconds
- Why is this a problem?
  - 55seconds < MSL!
- Use timestamp to distinguish sequence number wraparound