15-441 15-641 Computer Networking

Lecture 14 – TCP Performance & Future

Peter Steenkiste

Fall 2014

www.cs.cmu.edu/~prs/15-441-F14

Outline

- TCP status and extensions
  - Where are we
  - Why we need queues
  - Filling in the gaps
- TCP performance model
- Beyond basic TCP
  - TCP-friendly
  - Further optimizing performance

TCP so far

- Reliable byte stream protocol
- Connection establishments and tear down
  - Maintain state at end points to optimize performance
- Flow control to avoid flooding receiver
  - Based on sliding window to overcome RTT
- Error control to recover from lost packets
  - Cover up errors by best effort IP service
- Congestion control to avoid flooding the network
  - Protect the network – avoid congestion collapse

Error Control Has Evolved

- Original error control based on cumulative ACKs Go-Back-N
  - But only retransmit one packet to avoid wasting bandwidth
- Added fast retransmit - ~NACK
  - Try to avoid expensive timeout on packet loss
  - Not effective for bursty errors, small windows
- Selective ACK to avoid timeouts when using large windows
  - Multiple losses per window more common

Congestion Control also Evolved

- Original TCP did not have congestion control
  - Resulted in inefficiencies, congestion collapse
  - The price you pay for being successful!
- Congestion control based on implicit feedback
  - Binary: packet loss = congestion, no packet loss = OK
  - AIMD adaptation by sender – motivated by fairness
- Clever and scalable, but …
  - Routers need to drop packets to slow down sender
  - Crude, noisy feedback – more on this later
- Can we do better? Explicit feedback?

Outline

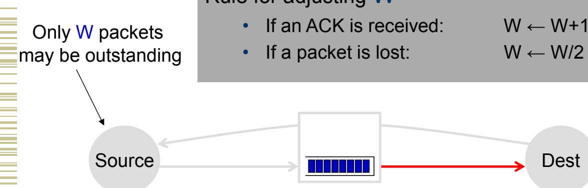
- TCP status and extensions
  - Where are we
  - Why we need queues
  - Filling in the gaps
- TCP performance model
- Beyond basic TCP
  - TCP-friendly
  - Further optimizing performance

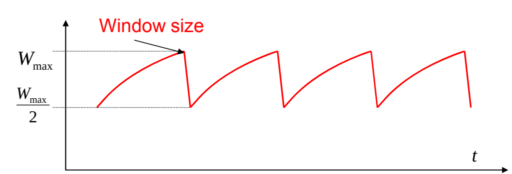
TCP Performance

- 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!

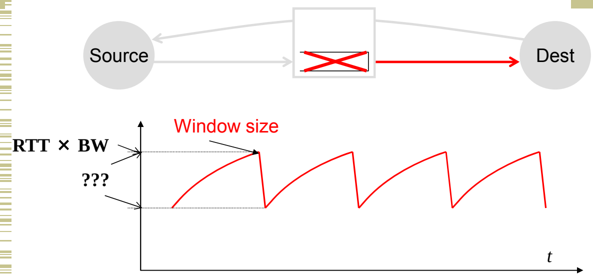
TCP Congestion Control

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





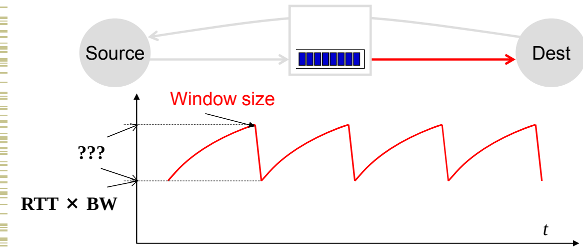
Single TCP Flow
Router without buffers



- Intuition: think in discrete time slots = RTT
- 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

9

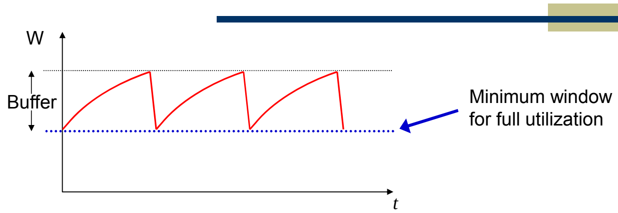
Single TCP Flow
Router with large enough buffers for full link utilization



- What is the minimum queue size for full utilization?
  - Must make sure that link is always full, even with smallest window
    - W/2 > RTT * BW - also W = 2 * RTT * BW = RTT * BW + Qsize
    - Therefore, Qsize > RTT * BW
  - Delay? Varies between RTT and 2 * RTT

10

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
    - I.e. when window is larger, buffering increases RTT
    - Buffer Size = Height of TCP Sawtooth
  - This is the origin of the rule-of-thumb
- Routers queues play critical role, not just to deal with burstiness of traffic, but also to allow TCP to fully utilize bottleneck links

But, at what cost!?

11

Outline

- TCP status and extensions
  - Where are we
  - Why we need queues
  - Filling in the gaps
- TCP performance model
- Beyond basic TCP
  - TCP-friendly
  - Further optimizing performance

12

How Was TCP Able to Evolve

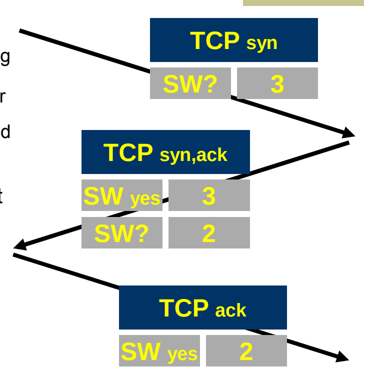
- Change endpoint behavior only
  - Fast retransmit, congestion control (implicit feedback)
- Use options to add information to the header
  - SACK – awkward but worth it; affects end point only
  - Example: window scaling
  - SYN cookies
  - Timestamp option
- Change the header!
  - Example: Explicit Congestion Notification

High Throughput Requires 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



Protection From Wraparound Timestamp option

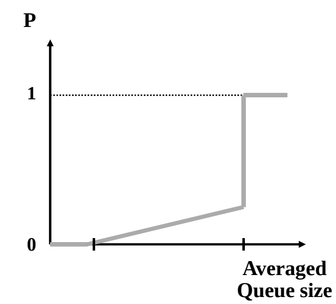
- Wraparound time vs. Link speed
  - 1.5Mbps: 6.4 hours
  - 10Mbps: 57 minutes
  - 45Mbps: 13 minutes
  - 100Mbps: 6 minutes
  - 622Mbps: 55 seconds
  - 1.2Gbps: 28 seconds
- Why is this a problem?
  - 55seconds < MSL!
- Use timestamp to distinguish sequence number wraparound

TCP Performance Issues

- Consistently full queues can degrade TCP performance.
  - Can lock out some sessions
  - Increased queueing delay
  - Lots of ongoing work on reducing buffer sizes
- Detection of congestion requires a packet loss – seem undesirable
- Bursts of packet losses can synchronize TCP sessions
  - All (many) sessions cut their window at the same time, they ramp up at the same time, …
  - Can lead to underutilization of the link
- Can we do better?

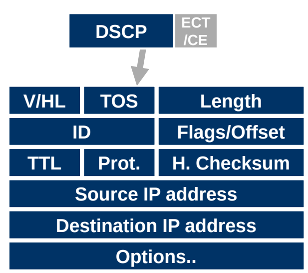
Random Early Detection (RED)

- Start randomly dropping packets before queue is full.
  - Some flows will observe a single packet loss and slow down, hopefully avoiding queue overflow
  - High bandwidth users are more likely to have a packet dropped than low bandwidth users
  - Queue can still accommodate bursts of packets
- Improves overall network performance by avoiding that queues stay full.
  - Congestion avoidance
  - How do you set the thresholds?



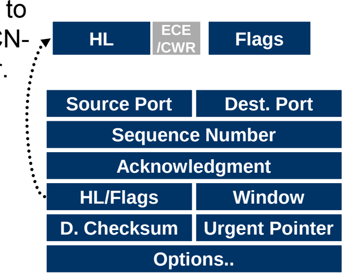
Explicit Congestion Notification (ECN)

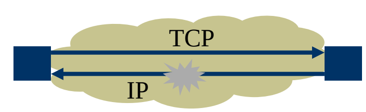
- The goal is to provide explicit congestion notification to senders
  - Complements the implicit feedback through packet drops
- Bits 6-7 of the TOS bit form the ECN field
  - The ECN-Capable Transport (ECT) bit is set by the sender to indicate that the end-points are ECN-capable
  - The Congestion Experience (CE) bit is set by the router to signal congestion
  - Reinterpreting bits in header a major obstacle to deployment!!!
- ECN is received by receiver, who must forward ECN info to the sender – how?



ECN in TCP

- Receiver signals congestion to the sender by setting the ECN-Echo flag in the TCP header.
  - Unused bit of the TCP header
  - Handles asymmetric routes
  - ECN-Echo flag also used to negotiate ECN use
- RED and ECN compliment each other
  - RED sets ECN bit – no loss!





And Now for the Really Messy Bits

- TCP uses delayed ACK: acks every other packet
  - Kind of messy interferes with: congestion control, fast retransmit (no delay), slow start, ….
- Nagle's algorithm avoids sending many small packets
  - Allow only one outstanding small (not full sized) segment that has not yet been acknowledged
  - Can be disabled for interactive applications (e.g., telnet)
- Silly window syndrome
  - If receiver advertises small increases in the receive window then the sender may waste time sending lots of small packets
  - Solution: don't do it – receiver tries to wait for one MSS
- Unusual circumstances: keep alive, RESET, …

Outline

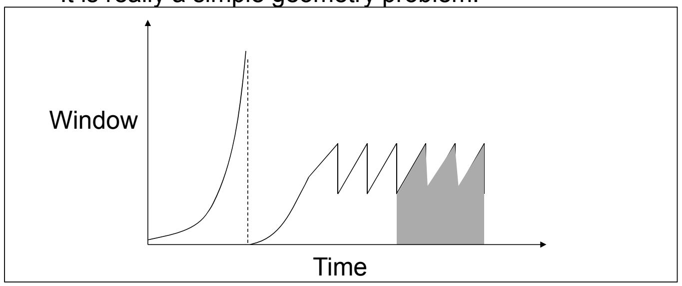
- TCP status and extensions
  - Where are we
  - Why we need queues
  - Filling in the gaps
- TCP performance model
- Beyond basic TCP
  - TCP-friendly
  - Further optimizing performance

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

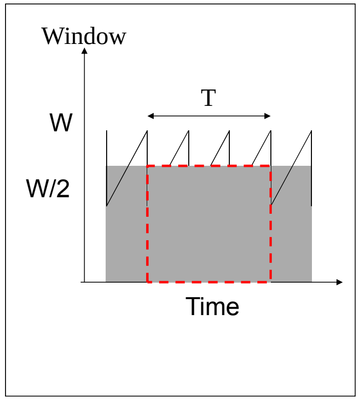
Overall TCP Behavior

- Let's concentrate on steady state behavior with no timeouts and perfect loss recovery
- Packets transferred = area under curve
  - It is really a simple geometry problem!



Transmission Rate

- What is area under curve?
  - Window in packets
  - Time in RTTs
  - A = avg window * time
    = ¾ W * T (packets)
- What was bandwidth?
  - BW = A / T = ¾ W
    - In packets per RTT
  - 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 loses a packet each time window reaches W packets
  - Window drops to W/2 packets
  - Each RTT window increases by 1 packet
    → W/2 * RTT between packet losses

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
  - $W = \sqrt{\frac{8}{3p}} = \frac{4}{3} \times \sqrt{\frac{3}{2p}}$
  - $BW = \frac{MSS}{RTT \times \sqrt{2p/3}}$

Throughput Equation Implication

- 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

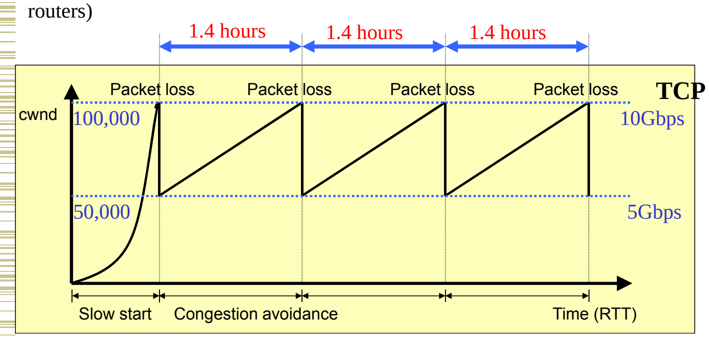
Throughput Equation Implication 2

$$T \approx \frac{\sqrt{1.5}\, MSS}{RTT\, \sqrt{p}}$$

- Suppose RTT = 100 ms, MSS = 1.5 KB
- T = 100 Gb/sec
- p=?
  - $p \approx 2 \times 10^{-12}$
- 1 drop every 6 petabits (17 hours).
- So….

TCP over High-Speed Networks

- A TCP connection with 1250-Byte packet size and 100ms RTT is running over a 10Gbps link (assuming no other connections, and no buffers at routers)



Source: Rhee, Xu. "Congestion Control on High-Speed Networks"

Outline

- TCP status and extensions
  - Where are we
  - Why we need queues
  - Filling in the gaps
- TCP performance model
- Beyond basic TCP
  - TCP-friendly
  - Further optimizing performance

Other Transport Protocols

- Request-response: overhead too high
  - Use RPC: response is ACK for request
  - What about congestion control?
- Multi-media streaming: timeouts add excessive delays, reducing "Quality of Experience"
  - Typically implement custom transport in the application
  - Can either tolerate some losses, use FEC, or use TCP, but avoid negative impact of timeouts
  - Real-time Transport Protocol can help
    - Limited scope: packet format, control information
  - What about congestion control?

TCP Friendliness

- What does it mean to be TCP friendly?
  - TCP is not going away any time soon
    - Although some people are working on it!
  - 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 a 1/sqrt(p) then the behavior it is ok
  - But is this really true?

Let's stop for a moment

- What can the network (really) do?
  - Enforce
    - Maximum aggregate rate & buffer (has to)
    - Isolation?
    - Fair sharing?
  - Inform
    - Aggregate limits exceeded (by packet drop)
    - Queue lengths (by delay) (or explicitly)
    - Degree of congestion?
    - Allowed rate?
- What are the end hosts' options?

TCP-Friendly Rate Control (TFRC)

- Goal: "like TCP, but smoother"
  - Avoid timeouts and dramatic rate changes
- Idea: calculate allowed rate using the TCP equation
  - Based on measured packet loss rate
  - Matches rate that TCP would have achieved
- Maintain smoothed estimate of loss rate, RTT
- Implement rate control through inter-packet time *t*

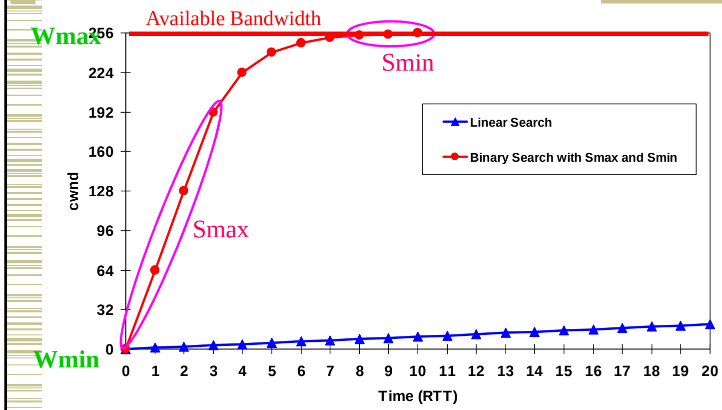
TCP Vegas

- Is it possible to avoid packet loss by observing the onset of congestion, i.e. proactive vs. reactive
- Idea: RTT changes with degree of congestion:
  - Minimal with empty buffers, longest right before drop
- Delay-based rate control (TCP Vegas)
  - Goal: Respond to congestion **before** buffers are full
  - Look at delay is a continuous feedback signal
- Estimate minimum (no-queuing) RTT and RTT with expected throughput
  - Reduce rate when throughput close to expected rate

TCP (CU)BIC

- Goal is to spend more time at the high end of the window value range
  - Remember: 1.4 hours to reach Wmax on 10 Gbs link?
- Idea: make the additive increase adaptive
  - Fast recovery toward Wmax
  - Slow change around (expected) Wmax
  - Fast search for (higher) Wmax

Binary Search with Smax and Smin



The TCP Reality

- Most file transfers are very small
  - TCP never reaches steady state – slow start dominates
- “TCP-fairness” is calculated on a per flow basis
  - Many browsers open parallel TCP sessions, oops
  - Other ways to cheat: what is your initial window?
- TLS is widely used – adds 1-2 RTT handshake
  - Starts after the TCP handshake!
- Motivates the design of new transport protocols
  - E.g., Google QUIC layered on top of UDP
- But there are many long flows as well!