15-441 15-641 Computer Networking

Lecture 14: TCP Performance & Future

Peter Steenkiste

Fall 2016

www.cs.cmu.edu/~prs/15-441-F16

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

Outline

- Evolution of TCP
  - Error and flow control
  - Random early detection
  - Explicit congestion notification
  - Window scaling
- TCP performance

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
  - Timestamp option
- Change the header!
  - Not very common
  - Example: Explicit Congestion Notification

Error Control Has Evolved

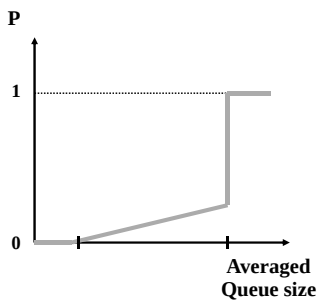
- Original error control based on cumulative ACKs Go-Back-N
  - But only retransmit one packet to avoid wasting bandwidth
- Added fast retransmit - ~NACK
  - Try to avoid expensive timeout on packet loss
  - Not effective for burst errors, small windows
- Selective ACK to avoid timeouts when using large 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
  - Packet drops can also synchronize TCP senders
- Can we do better? Explicit feedback?

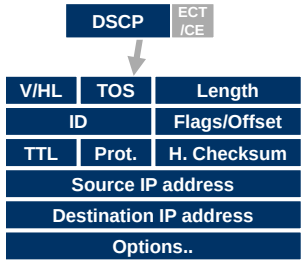
Random Early Detection (RED)

- 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!!!



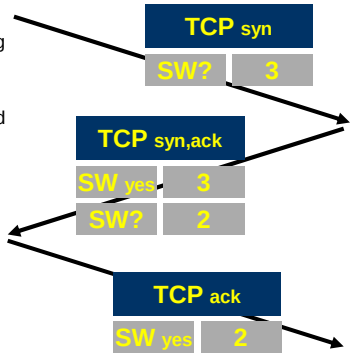
- ECN is received by receiver, but needed by sender – How?
  - ECN bit is carried to sender in the TCP header (flags field)

Evolution of Flow Control: 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



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 syndrom
  - 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, …

The TCP Reality

- Most file transfers are very small (mice)
  - TCP never reaches steady state – slow start dominates
  - Fast retransmit often fails – very bad for latency
- But there are long flows as well! (elephants)
  - Next
- "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!

Outline

- Evolution of TCP
  - Error and flow control
  - Random early detection
  - Explicit congestion notification
  - Window scaling
- TCP performance

TCP Modeling

- Given the congestion behavior of TCP can we predict what type of performance we should get?
  - Assume no timeouts and perfect error recovery
- What are the important factors
  - Loss rate: Affects how often window is reduced
  - RTT: Affects increase rate and relates BW to window
  - MSS: Affects increase rate (additive increase)
- Result:

$$BW = \frac{MSS}{RTT \times \sqrt{2p/3}}$$

Implication 1: Fairness

- Flows sharing bottleneck do NOT get same bandwidth!
  - Fairness: BW proportional to 1/RTT
- TCP is “RTT fair”
  - Only “if all else is equal” do flows sharing a bottleneck get the same bandwidth
  - Not by design

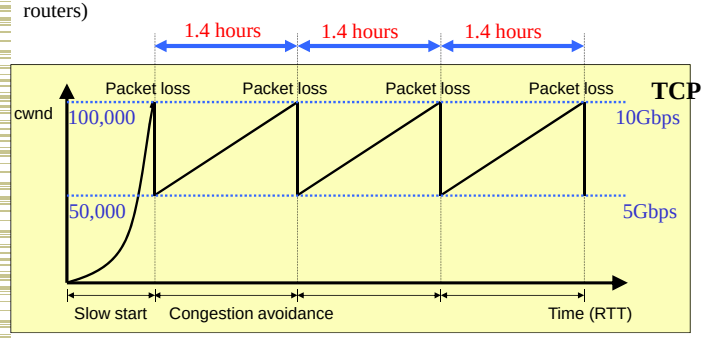
Implication 2: High Speed Networks

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

- 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)

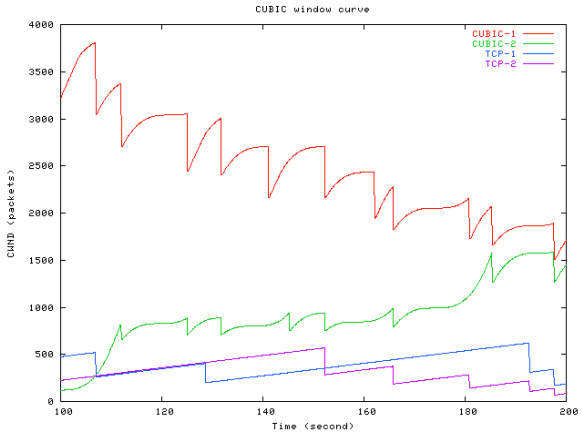


Source: Rhee, Xu. "Congestion Control on High-Speed Networks"

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 depending on how close you are to presumed Wmax value
  - Fast recovery using larger additive increase toward Wmax (cubic increase)
  - Slow change around Wmax
  - Fast search for a higher Wmax

TCP CUBIC in one slide



Implication 3: How about non-TCP flow?

- Certain types of flow cannot use TCP
  - E.g., multi-media streaming: timeouts add excessive delays
  - Require custom transport, but what congestion control?
- Solution TCP: make them "TCP friendly"
  - "Like TCP but smoother"
  - Motivation is that they need to coexist nicely with TCP
  - Should hold their own without clobbering TCP flows
- Their throughput must follow the TCP equation
  - Calculated smoothed estimates of RTT, p, …
  - Do TCP-Friendly Rate Control (TFRC) based on TCP equation
  - Can adjust the transmit rate without timeouts

Optional Material

Derivation of
Throughput Model

21

TCP Modeling

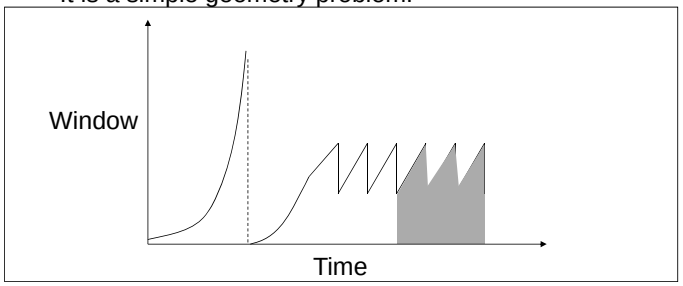
- Given the congestion behavior of TCP can we predict what type of performance we should get?
  - Assume no timeouts and perfect error recovery
- What are the important factors
  - Loss rate: Affects how often window is reduced
  - RTT: Affects increase rate and relates BW to window
  - MSS: Affects increase rate (additive increase)
- Result:

$$BW = \frac{MSS}{RTT \times \sqrt{2p/3}}$$

22

Overall TCP Behavior

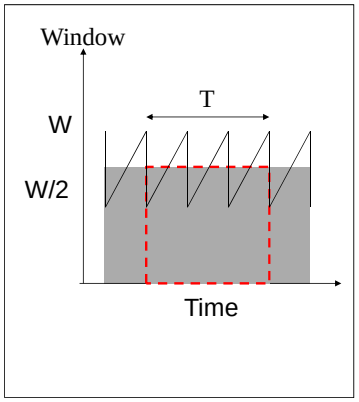
- Let us concentrate on steady state behavior with no timeouts and perfect loss recovery
- Packets transferred = area under curve
  - It is a simple geometry problem!



23

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



24

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