Midterm score distribution

TCP congestion control: additive increase, multiplicative decrease

Approach: increase transmission rate (window size), probing for usable bandwidth, until loss occurs
- Additive increase: increase CongWin by 1 MSS every RTT until loss detected
- Multiplicative decrease: cut CongWin in half after loss

Saw tooth behavior: probing for bandwidth

TCP Congestion Control: details
- Sender limits transmission:
  \[ \text{CongWin} = \text{LastByteSent} - \text{LastByteAcked} \]
- Roughly,
  \[ \text{rate} = \frac{\text{CongWin}}{\text{RTT}} \text{ Bytes/sec} \]
- CongWin is dynamic, function of perceived network congestion
- How does sender perceive congestion?
  - Loss event = timeout or 3 duplicate acks
  - TCP sender reduces rate (CongWin) after loss event
- Three mechanisms:
  - AIMD
  - Slow start
  - Conservative after timeout events

TCP Slow Start
- When connection begins, CongWin = 1 MSS
  - Example: MSS = 500 bytes & RTT = 200 msec
  - Initial rate = 20 kbps
- Available bandwidth may be >> MSS/RTT
  - Desirable to quickly ramp up to respectable rate
  - When connection begins, increase rate exponentially until first loss event:
    - Double CongWin every RTT
    - Done by incrementing CongWin for every ACK received
- Summary: initial rate is slow but ramps up exponentially fast

Refinement
- Q: When should the exponential increase switch to linear?
  - A: When CongWin gets to 1/2 of its value before timeout.
- Implementation:
  - Variable Threshold
  - At loss event, Threshold is set to 1/2 of CongWin just before loss event
**Refinement: inferring loss**

- After 3 dup ACKs:
  - CongWin is cut in half
  - Window then grows linearly
- But after timeout event:
  - CongWin set to 1 MSS
  - Window then grows exponentially
  - To a threshold, then grows linearly

**Philosophy:**

3 dup ACKs indicates network capable of delivering some segments
Timeout indicates a "more alarming" congestion scenario

**Summary: TCP Congestion Control**

- When CongWin is below Threshold, sender in slow-start phase, window grows exponentially.
- When CongWin is above Threshold, sender is in congestion-avoidance phase, window grows linearly.
- When a triple duplicate ACK occurs, Threshold set to CongWin/2 and CongWin is set to Threshold.
- When timeout occurs, Threshold set to CongWin/2 and CongWin is set to 1 MSS.

**TCP sender congestion control**

<table>
<thead>
<tr>
<th>State</th>
<th>Event</th>
<th>TCP Sender Action</th>
<th>Commentary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Slow Start (SS)</td>
<td>ACK receipt for previously unacked data</td>
<td>CongWin = CongWin + MSS, If (CongWin &gt; Threshold) set state to “Congestion Avoidance”</td>
<td>Resulting in a doubling of CongWin every RTT</td>
</tr>
<tr>
<td>Congestion Avoidance (CA)</td>
<td>ACK receipt for previously unacked data</td>
<td>CongWin = CongWin + MSS * (MSS/CongWin)</td>
<td>Additive increase, resulting in increase of CongWin by 1 MSS every RTT</td>
</tr>
<tr>
<td>SS or CA</td>
<td>Loss event detected by triple duplicate ACK</td>
<td>Threshold = CongWin/2, CongWin = 1 MSS, Set state to “Slow Start”</td>
<td>Fast recovery, implementing multiplicative decrease. CongWin will not drop below 1 MSS.</td>
</tr>
<tr>
<td>SS or CA</td>
<td>Timeout</td>
<td>Threshold = CongWin/2, CongWin = 1 MSS, Set state to “Slow Start”</td>
<td>Enter slow start</td>
</tr>
<tr>
<td>SS or CA</td>
<td>Duplicate ACK</td>
<td>Increment duplicate ACK count for segment being asked</td>
<td>CongWin and Threshold not changed</td>
</tr>
</tbody>
</table>

**TCP Throughput**

- What’s the average throughout of TCP as a function of window size and RTT?
- Ignore slow start
- Let W be the window size when loss occurs.
- When window is W, throughput is W/RTT
- Just after loss, window drops to W/2, throughput to W/2RTT.
- Average throughout: .75 W/RTT

**TCP Futures**

- Example: 1500 byte segments, 100ms RTT, want 10 Gbps throughput
- Requires window size W = 83,333 in-flight segments
- Throughput in terms of loss rate:

\[
\frac{1.22 \times MSS}{RTT \sqrt{L}}
\]

\[\Rightarrow L = 2 \times 10^{30} \text{ Wow} \]

- New versions of TCP for high-speed needed!
Why is TCP fair?

Two competing sessions:
- Additive increase gives slope of 1, as throughput increases
- Multiplicative decrease decreases throughput proportionally

Fairness and UDP
- Multimedia apps often do not use TCP
  - Do not want rate throttled by congestion control
- Instead use UDP:
  - Pump audio/video at constant rate, tolerate packet loss
- Research area: TCP friendly

Fairness and parallel TCP connections
- Nothing prevents app from opening parallel connections between 2 hosts.
- Web browsers do this
  - Example: link of rate R supporting 9 connections:
    - New app asks for 1 TCP, gets rate R/10
    - New app asks for 11 TCPs, gets R/2!