Chapter 3
Transport Layer

A note on the use of these ppt slides:
We’re making these slides freely available to all (faculty, students, readers). They’re in PowerPoint form so you can add, modify, and delete slides (including this one) and slide content to suit your needs. They obviously represent a lot of work on our part. In return for use, we only ask the following:
- If you use these slides (e.g., in a class) in substantially unaltered form, that you mention their source (after all, we’d like people to use our book!)
- If you post any slides in substantially unaltered form on a www site, that you note that they are adapted from (or perhaps identical to) our slides, and note our copyright of this material.

Thanks and enjoy! JFK/KWR

Computer Networking: A Top Down Approach
Jim Kurose, Keith Ross
Addison-Wesley, July 2007.

All material copyright 1996-2007
J.F Kurose and K.W. Ross, All Rights Reserved
Chapter 3 outline

- 3.1 Transport-layer services
- 3.2 Multiplexing and demultiplexing
- 3.3 Connectionless transport: UDP
- 3.4 Principles of reliable data transfer
- 3.5 Connection-oriented transport: TCP
  - segment structure
  - reliable data transfer
  - flow control
  - connection management
- 3.6 Principles of congestion control
- 3.7 TCP congestion control
TCP congestion control: additive increase, multiplicative decrease

- **Approach**: increase transmission rate (window size), probing for usable bandwidth, until loss occurs
  - **additive increase**: increase \( \text{CongWin} \) by 1 MSS every RTT until loss detected
  - **multiplicative decrease**: cut \( \text{CongWin} \) in half after loss

Saw tooth behavior: probing for bandwidth
TCP Congestion Control: details

- sender limits transmission: $\text{LastByteSent} - \text{LastByteAcked} \leq \text{CongWin}$

- Roughly, $\text{rate} = \frac{\text{CongWin}}{\text{RTT}}$ 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, $\text{CongWin} = 1 \text{ MSS}$
  - Example: $\text{MSS} = 500$ bytes & $\text{RTT} = 200$ msec
  - initial rate $= 20$ kbps

- available bandwidth may be $>> \text{MSS/RTT}$
  - desirable to quickly ramp up to respectable rate

- When connection begins, increase rate exponentially fast until first loss event
TCP Slow Start (more)

- 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 instead 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 is 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 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 = Threshold, Set state to “Congestion Avoidance”</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 acked</td>
<td>CongWin and Threshold not changed</td>
</tr>
</tbody>
</table>

- **Fast Recovery:**
  a) duplicate ACK: CongWin = CongWin + MSS and keep FR
  b) new ACK → Congestion Avoidance, but CongWin drops to Threshold
  c) timeout → Transition to Slow Start, exactly as in other cases