Overview: TCP Basics

- Recap: using SEQ and ACK numbers
  - SEQ random initial number for numbering the bytes in the application message
  - “ACKnowledge next byte expected
  - Read chapter to review
- TCP congestion control
  - Read through multiple times!
- TCP flow control
TCP: SEQ and ACK numbers

TCP: Cumulative ACK

Cumulative ACK scenario
TCP possible sender events:

(1) **Data received from application:**
1. Create a segment and assign a SEQ number
   - SEQ # is byte-stream number of first data byte in segment
2. Start timer if it is not already running
   - Timer is for the oldest un-acked segment
   - Expiration interval: TimeoutInterval

(2) **Timeout (ACK not received):**
1. Retransmit segment that caused the timeout
2. Restart the timer

(3) **ACK received for previously unacked segments**
1. Update what is known to be acked
2. Start timer if there are outstanding segments

TCP: retransmission scenarios

1) What is/was ‘A’s next step?
2) What does ‘B’ then do?
What does ‘A’ do next, and when does it do it?

New Today:
Principles of Congestion Control

- Packet loss is caused by overflowing router buffers
- Retransmission treats the symptom
- Congestion control treats the cause
- What are costs of congestion?
TCP Congestion Control

- Three questions
  1. How does a sender sense congestion?
  2. How does a sender limit its sending rate?
  3. What algorithm is used to change the send-rate?

TCP Congestion Control: details

How does sender perceive congestion?
- A loss event is?
  - A timeout or
  - 3 duplicate ACKs

How does sender limit its send rate?
- TCP sender reduces the send rate via changing a variable value, the "CongWin," after a loss event
  \[ \text{LastByteSent} - \text{LastByteAcked} \leq \text{CongWin} \]
**TCP Congestion Control Window**

- CongWin is dynamic, function of perceived network congestion
- Sender limits transmission:
  \[ \text{LastByteSent} - \text{LastByteAcked} \leq \text{CongWin} \]
- Changing CongWin changes the SendRate

---

**TCP Congestion Control Algorithm**
TCP Congestion Control Algorithm

Three major mechanisms:

1) Slow start
2) Congestion Avoidance
   - AIMD = additive increase, multiplicative decrease
3) Fast Recovery:
   - Reaction to timeout events versus 3 duplicate ACKs

Are used to adjust...?

- Adjust CongWin

TCP Slow Start

1) When connection begins, CongWin = 1 MSS
   - Available bandwidth probably much greater
   - Desirable to quickly ramp up to respectable rate
2) Increase rate exponentially fast until first loss event
   - Grow window 1 MSS for each ACK received

Summary: initial rate is slow but ramps up exponentially fast
TCP: detecting, reacting to loss

- Loss indicated by timeout
  - \textit{cwnd} set to 1 MSS;
  - Window (\textit{cwnd}) grows exponentially (slow start) to the threshold, then grows linearly

- Loss indicated by 3 duplicate ACKs
  - Duplicate ACKs indicate network capable of delivering some segments
  - \textit{cwnd} is cut in half window then grows linearly

TCP Slow Start with Exp. Increase

- When connection begins, increase rate exponentially until first loss event:
  - Double CongWin every RTT
  - Increment CongWin for every ACK received
**TCP: Congestion algorithm switching from slow start to CA**

**Implementation:**
- variable `ssthresh`
- on loss event, `ssthresh` is set to 1/2 of `cwnd` just before loss event

---

**Reaction to Loss Events**

- Exponential increase switches to linear increase when `CongWin` gets to the ‘threshold’ value (size)
Summary: TCP Congestion Control

Increase Sending Rate Phase Options:
1. When $\text{CongWin}$ is below $\text{Threshold}$, sender in slow-start phase, window grows exponentially.
2. When $\text{CongWin}$ is above $\text{Threshold}$, sender is in congestion-avoidance phase, window grows linearly.

Decrease Sending Rate Phase Options:
1. When a triple duplicate $\text{ACK}$ occurs, Threshold set to $\text{CongWin}/2$ and $\text{CongWin}$ set to $\text{Threshold}$.
2. When timeout occurs, Threshold set to $\text{CongWin}/2$ and $\text{CongWin}$ is set to 1 MSS.