TCP Congestion Control (Continued)

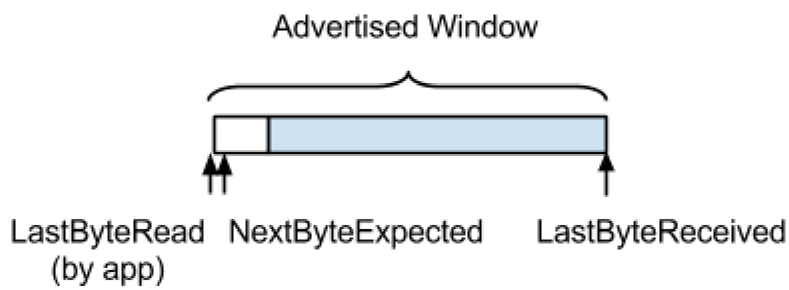
CS640, 2015-03-19

Outline

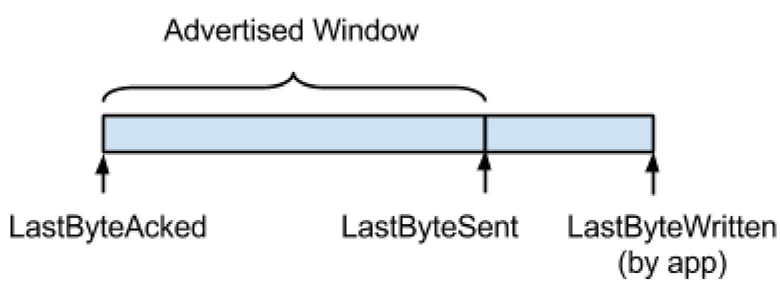
- Fast retransmit/recovery

Fast Retransmit/Fast Recovery

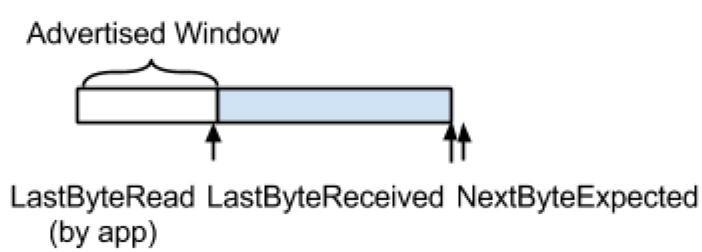
- Problem: waiting for timeout when loss occurs results in a long time period where no packets are sent
  - No packets are sent because effective window becomes 0
    - Receiver buffer



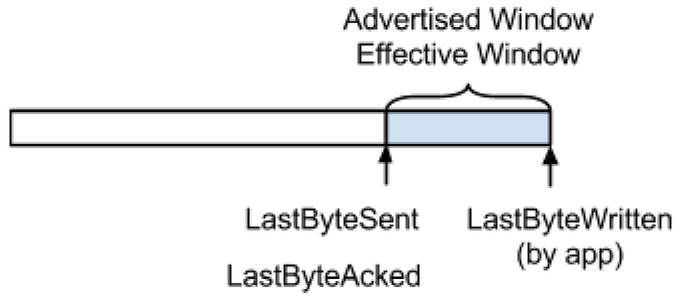
    - Sender buffer



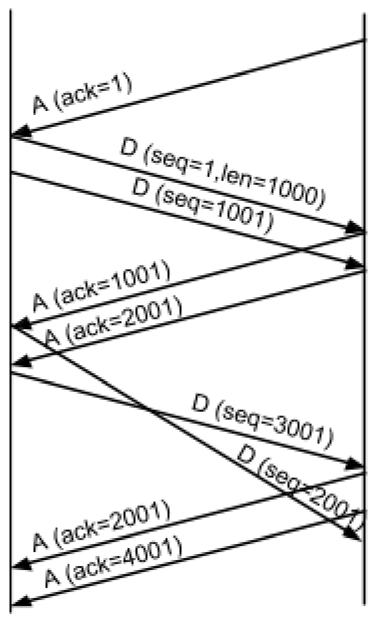
  - Once lost data is retransmitted and received, application can consume data, which will cause sliding window to shift and effective window to become non-zero
    - Receiver buffer



■ Sender buffer

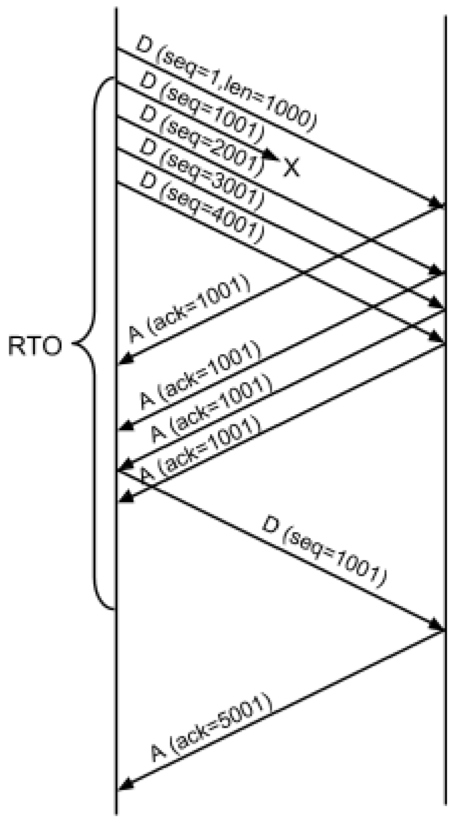


- Fast retransmit
  - Timeout is not the only sign of loss
  - Receiving multiple ACKs with the same sequence number could indicate:
    - Data is out of order and earlier data has not yet arrived



    - Data has been lost (i.e., earlier data will never arrive unless retransmitted)

- Treat three duplicate ACKs (i.e., ACKs with same ACK #) as sign of loss



  - Retransmit packet starting with seq # contained in duplicate ACKs
  - Allows for moderate amount of re-ordering -- re-ordering is relatively rare
  - No longer need to wait for timeout before loss is detected and recovered -- means more data can be sent sooner

- Fast recovery
    - Avoid running slow start every time a loss occurs -- it takes too long to ramp up when bandwidth is high
    - When fast retransmit occurs
        - Set congestion window (CWND) to slow start threshold (SSTHRESH) when loss occurs
        - Then do additive increase
    - Use slow start only at connection start or when timeout occurs

Congestion Avoidance

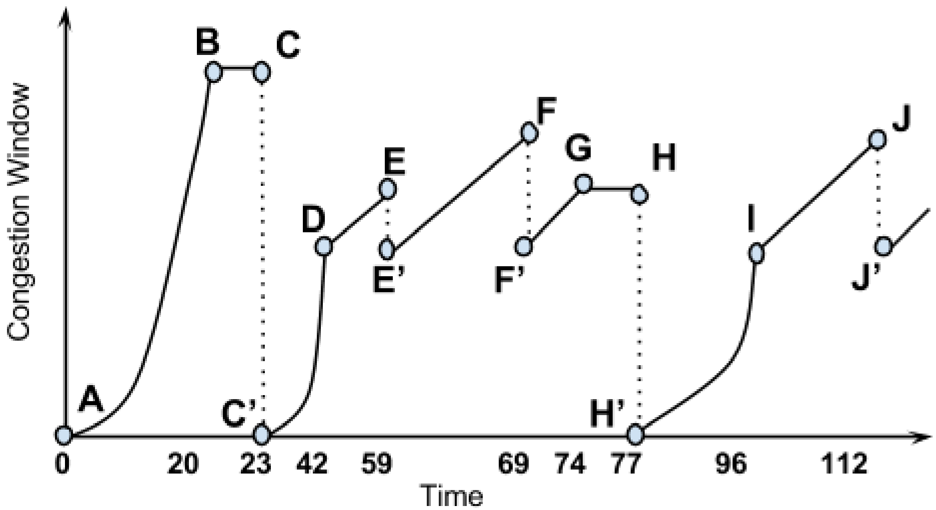
- Do not wait until loss occurs; watch for signs of emerging congestion
- *****What can we use as a sign of emerging congestion? Think about what causes congestion.***
    - As packet queues build up in routers, there is a measurable increase in RTT for each successive packet
- Approach #1: compare current RTT to average RTT
    - Every two RTTs check if current RTT > average(minimum RTT, maximum RTT)
    - If so, decrease CWND by 1/8th
- Approach #2: compare throughputs
    - At beginning of connection, measure throughput when one packet is in transit
        - Divide MSS by RTT

  Calculate throughput when window is of size CWND
        - Divide number of outstanding bytes by RTT
    - Increase CWND by one MSS
    - Calculate throughput when window is one MSS larger
    - Subtract throughput with CWND+1 from throughput with CWND
    - If throughput difference < ½ throughput measured at connection start (when one packet was in transit), then decrease CWND by 1

TCP Variants

- Tahoe -- exponential backoff + slow start + fast retransmit (no fast recovery)
- Reno -- Tahoe + fast recovery + delayed ACKs (Linux & OS X)
- Vegas -- Tahoe + congestion avoidance
- Others: cubic (Linux), westwood, bic, new reno (OS X), compound (Windows)

Example



a. Identify all time intervals when the TCP flow is undergoing slow start.
   A to B; C' to D; H' to I
b. Identify all points where a timeout occurs.
   C; H
c. Identify all points where fast recovery occurs.
   E; F; J
d. Calculate the value of the CWND at all labeled points.
   A = 1
   B = 2^{20}
   C = 2^{20}
   C' = 1
   D = 2^{19}
   E = $2^{19} + 17$
   E' = 2^{19}
   F = $2^{19} + 10$
   F' = 2^{19}
   G = $2^{19} + 5$
   H = $2^{19} + 5$
   H' = 1
   I = $(2^{19} + 5)/2$
   J = $(2^{19} + 5)/2 + 16$
   J' = $(2^{19} + 5)/2$
e. Calculate the value of SSTHRESH at all labeled points.
   A = ∞
   B = ∞
   C = ∞
   C' = 2^{19}
   D = 2^{19}
   E = 2^{19}
   E' = 2^{19}
   F = 2^{19}

$F' = 2^{19}$
$G = 2^{19}$
$H = 2^{19}$
$H' = (2^{19} + 5)/2$
$I = (2^{19} + 5)/2$
$J = (2^{19} + 5)/2$
$J' = (2^{19} + 5)/2$