Lecture 16 (March 23, 2004)

Outline
- New Reno
- SACK
- TCP Vegas
- RED

Motivation for improving RENO
- Recall from Internet traffic lecture that packets are often transmitted in bursts
- Thus, losses also often happen in bursts
  - Due to FIFO (drop tail) queues in routers
- Problem with TCP RENO: If new ACK after Fast Retransmit is not for all data transferred since dup ACK, then DO NOT do Fast Recovery
  - Since only mechanism for recognizing other drops is a timeout

TCP New Reno
- Simple observation…
- Recognized that losses happen in bursts
- Upon receipt of a partial ACK (ACK for some but not all outstanding data), retransmit next data packet in sequence
- Same congestion control mechanism as RENO
  - But, do not avoid Fast Recovery on multiple drops
- Enables recovery from multiple packet losses in sequence without timeout
  - Timeouts are still used…

TCP SACK
- Guess what? Losses do not always happen in bursts – sometimes things get a little crazy
  - How about telling the sender what has arrived (1)
- Selective Acknowledgements (SACK)
  - Same congestion control mechanisms as TCP RENO
  - Uses TCP options fields
  - When out of order data arrives, tell sender (via SACK block) which block of data has been received
  - Enables sender to maintain image of receiver’s queue
  - Sender then resends all missing segments without waiting to timeout
  - Doesn’t send beyond CWND
  - When no old data needs to be resent, then send new data
  - Timeouts are still used…

Congestion Avoidance
- TCP’s strategy
  - Control congestion once it happens
  - Generally increase load in an effort to find the point at which congestion occurs, and then back off
- Alternative strategy
  - Predict when congestion is about to happen
  - Reduce rate before packets start being discarded
  - Call this congestion avoidance, instead of congestion control
- Two possibilities
  - Host-centric: TCP Vegas
  - Router-centric: RED Gateways

TCP Vegas
- Idea: source watches for some sign that router’s queue is building up and congestion will happen too; e.g.,
  - RTT grows
  - Sending rate flattens
  - Buffer space at router
  - In shaded region we expect throughput to increase but it cannot increase beyond available bandwidth
### Vegas Algorithm

- Vegas tries not to send at a rate that causes buffers to fill
- Let BaseRTT be the minimum of all measured RTTs
- If not overflowing the connection, then
  \[ \text{ExpectRate} = \frac{\text{CongestionWindow}}{\text{BaseRTT}} \]
  Assume CWND = number of bytes in transit
- Source calculates sending rate (ActualRate) once per RTT
  - Pick one packet per RTT, timestamp send/ACK receive time and
divides by number of bytes in transit
- Source compares ActualRate with ExpectRate
  - If Diff = ExpectRate - ActualRate
    - increase CongestionWindow linearly
  - else if Diff > α
    - decrease CongestionWindow linearly
  - Else
    - leave CongestionWindow unchanged

### Algorithm (cont)

- Parameters
  - \( \alpha = 1 \) buffer
  - \( \beta = 3 \) buffers

### Random Early Detection (RED)

- Congestion notification is implicit
  - dropped packets indicate congestion
- Could make explicit by marking the packet (ECN)
  - Current standard proposal for explicit notification (old method: DECbit)
  - Mark packet as queue gets full, reduce sending rate when marks are seen
- Early random drop to force sources to back off
  - rather than wait for queue to become full, drop each arriving packet with some drop probability whenever the queue length exceeds some drop level

### RED Details

- Compute average queue length
  \[ \text{AvgLen} = (1 - \text{Weight}) \times \text{AvgLen} + \text{SampleLen} \times \text{Weight} \]
  \( 0 < \text{Weight} < 1 \) (usually 0.002)
  SampleLen is queue length each time a packet arrives
  (Exactly the same as the EWMA calculation for RTT)

### RED Details (cont)

- Two queue length thresholds
  - if AvgLen \(<=\) MinThreshold then
    enqueue the packet
  - if MinThreshold \(<=\) AvgLen \(<=\) MaxThreshold then
    calculate probability \( P \)
    drop arriving packet with probability \( P \)
  - if MaxThreshold \(<=\) AvgLen then
    drop arriving packet

- Drop probability is a function of both AvgLen and how long it has been since the last drop.
  - TempP tracks how many newly arriving packets have been queued while AvgLen is between thresholds
  - Count is number of packets since last drop
  - This prevents clusters of drops
RED (contd)

- Thresholds are *hard* to determine
- Drop Probability Curve

- RED is good at keeping avg. queue size steady