## Reliable Data Transport

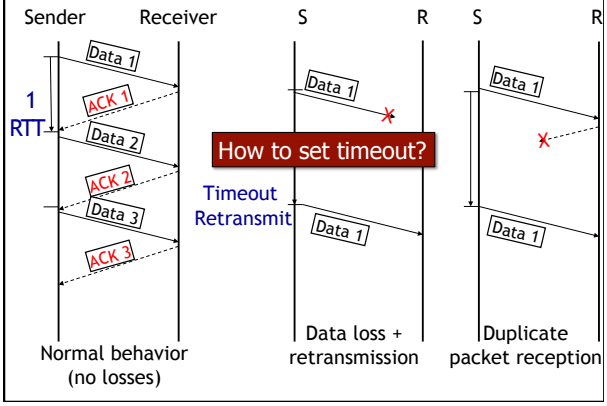
Lecture 20
6.02 Fall 2010
November 17, 2010

- Redundancy via careful retransmission
  - Sequence numbers and acknowledgments
  - RTT estimation and timeouts
- Stop-and-wait protocol
- Sliding window protocol

## The Problem

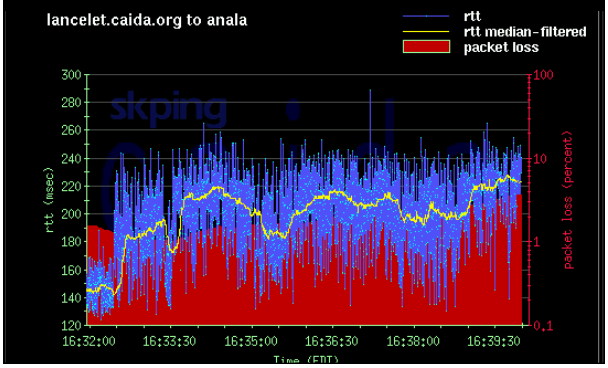
- Given: Best-effort network in which
  - Packets may be lost arbitrarily
  - Packets may be reordered arbitrarily
  - Packet delays are variable (queueing)
  - Packets may even be duplicated
- Sender S and receiver R want to communicate reliably
  - Application at R wants *all* data bytes in exactly the same order that S sent them
  - Each byte must be delivered exactly once
- Functions are provided by a reliable transport protocol
  - Application "layered above" transport protocol

## Stop-and-Wait Protocol



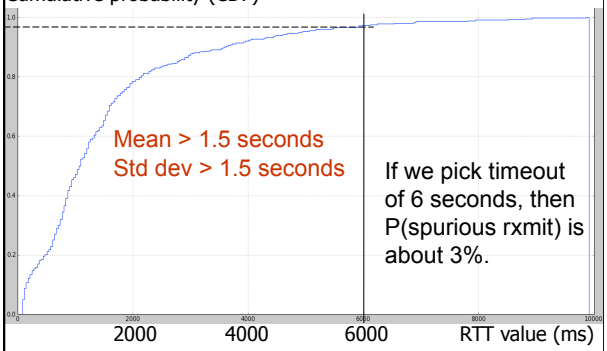

## RTT Measurements

(collected by CAIDA)

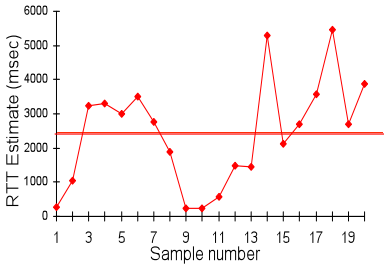



## CDF of RTT over Verizon Wireless 3G Network

Cumulative probability (CDF)



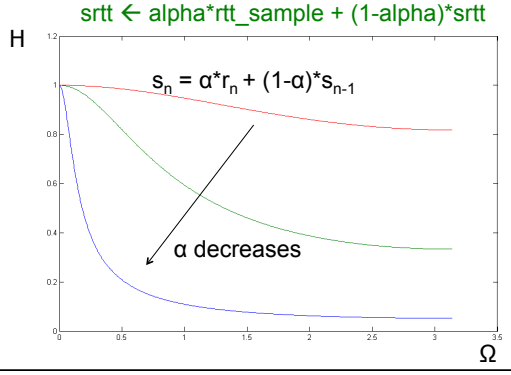

## Round-Trip Time (RTT) Could Be Highly Variable




Example from a TCP connection over a wide-area wireless link
Mean RTT = 2.4 seconds; Std deviation = 1.5 seconds!

## Exponential Weighted Moving Average (EWMA) LPF Frequency Response

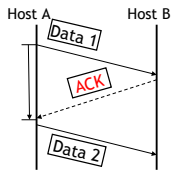
srtt ← alpha*rtt_sample + (1-alpha)*srtt




## Timeout Algorithm

- EWMA for smoothed RTT (srtt)
  - srtt ← alpha*rtt_sample + (1-alpha)*srtt
- Use another EWMA for RTT deviation (rttdev)
  - Mean linear deviation easy to compute (but could also do std deviation)
  - rttdev ← beta*dev_sample+ (1-beta)*rttdev, where dev_sample = |rtt_sample - srtt|
- Rxmit Timeout, RTO ← srtt + k*rttdev
  - k = 4 for TCP
  - Makes the "tail probability" of a spurious retransmission low

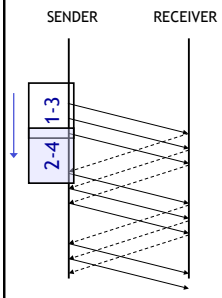
## Improving Performance



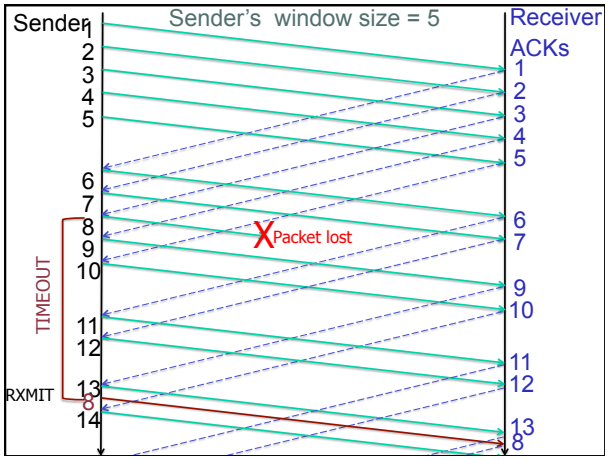

- Stop-and-wait protocol too slow
- Throughput = 1 packet per RTT
- 1500 byte pkt, 100 ms RTT throughput pegged at 15 KBytes/s
- With packet loss & timeout, throughput even lower

- Solution: Use a *window*
  - Keep multiple packets outstanding in the network at once
  - Overlap transmissions with ACKs

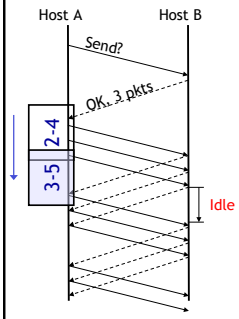
## Solution: Use a Sliding Window




- Senders advances the window by 1 for each in-sequence ack it receives
  - I.e., window *slides*
  - So, idle period reduces
  - **Pipelining**
- Assume that the window size, W, is fixed and known
  - Later, we will discuss how one might set it
  - W = 3 in the example on the left




## Setting the Window Size: Apply Little's Law




- If we can get "Idle" to 0, will achieve goal
- "N" = #packets in window
- C = rate
- RTT = avg delay
- If *W* = *C*RTT, path will be fully utilized
  - The **"bandwidth-delay product"**
  - Key concept in transport protocols