

## 6.02 – Network: Reliable Transport and Congestion Control

### Lecture #10

Katrina LaCurts, lacurts@mit.edu

#### 0. Introduction

- This lecture (+ the next, sort of): moving up one layer to talk about reliable transport

#### 1. TCP

- Goals: provide reliable transport, prevent congestion
- Broader questions: how do we do this scalably, and how do we share the network efficiently and fairly?
- Today: TCP Congestion Control
  - In particular, a version of TCP known as "New Reno"
- Next lecture: An alternative approach to "resource management" on the Internet

#### 2. Reliable Transport via sliding-window protocol

- Goal: receiving application gets a complete, in-order bytestream from the sender. One copy of every packet, in order.
- Why do we need it? Network is unreliable. Packets get dropped, can arrive out-of-order.
- Basics:
  - Every data packet gets a sequence number (1, 2, 3, ...)
  - Sender has  $W$  outstanding packets at any given time.  $W$  = window size
  - When receive gets a packet, it sends an ACK back. ACKs are cumulative: An ACK for  $X$  indicates "I have received all packets up to and including  $X$ ."
  - If sender doesn't receive an ACK indicating that packet  $X$  has been received, after some amount of time it will "timeout" and retransmit  $X$ .
    - Maybe  $X$  was lost, its ACK was lost, or its ACK is delayed
    - The timeout = proportional to (but a bit larger than) the RTT of the path between sender and receiver
  - At receiver: keep buffer to avoid delivering out-of-order packets, keep track of last-packet-delivered to avoid delivering duplicates.

#### 3. Main motivation

- What's the "right" value for  $W$ ?
- In particular, what if there are multiple senders? Perhaps we aim for them to share the bottleneck link equally, but how do  $S_1$  and  $S_2$  figure this out? What happens if  $S_3$  arrives? Or if  $S_1$  starts sending less? Etc.

#### 5. Congestion Control: controlling the source rate to achieve high performance

- Goals: Efficiency and fairness
  - Minimize drops, minimize delay, maximize utilization

- Share bandwidth fairly among all connections that are using it
- FOR NOW: assume all senders have infinite offered load. Fairness = splitting bandwidth equally amongst them.
- But no sender knows how many other senders there are, and that number can change over time.
- We'll use window-based congestion control. Switches are dumb (can only drop packets); senders are smart

## 6. AIMD

- Need a signal for congestion in the network, so senders can react to it.
- Our signal: packet drops
- Every RTT:
  - If there is no loss,  $W = W+1$
  - If there is loss,  $W = W/2$
- This is "Additive Increase Multiplicative Decrease" (AIMD)
- Senders constantly readjust => adapt to a changing number of senders, or changing offered loads
- Window size exhibits sawtooth behavior (see slides)
- Why AIMD?
  - It's "safe": senders are conservative about increasing, but scale back dramatically in the face of congestion
  - Efficient and fair

## 7. Finite Offered Load

- Remove the assumption that everyone has infinite offered load
- Suppose S1 and S2 have offered load of 10 packets/s, S3 has offered load of 5 packets/s, and they all share a bottleneck with capacity 20 packets/s. What happens?
  - In theory: S3 stops increasing once it's sending 5 packets/s. S1 and S2 continue increasing until they reach 7-8 packets/s.
- Is this fair?
  - In some sense. It achieves a type of fairness known as "max-min fairness". But there are other definitions (e.g., "proportional fairness")
- What happens in practice?
  - We might get max-min fairness, or one of the senders might experience a much longer RTT and so not increase its window at the same rate.
- So: TCP's congestion control utilizes the network reasonably well, but it's hard to measure fairness, or claim that fairness is achieved under skewed workloads, varying RTTs, etc.

## 8. Additional Mechanisms

- Slow Start
  - At the beginning of the connection, exponential increase the window (double it every RTT until you see loss)
  - Decreases the time it takes for the initial window to "ramp up"
  - (See slide for diagram)
- Fast Retransmit/Fast Recovery

- When a sender receives an ACK with sequence number X, and then three duplicates of that packet, it immediately retransmits packet X+1 (remember: ACKs are cumulative)

Ex: Send 1 2 3 4 5 6

Receive 1 2 2 2 2

Sender receives 4 ACKs total with sequence number "2"; infers that packet 3 is lost, immediately retransmits

- On fast-retransmit, window decrease is as before:  $W = W/2$
- In fact, when a packet is lost due to timeout, TCP behaves differently:  $W = 1$ , then do slow-start until the last good window, and then start additive increase.
- (See slide for diagram)
- Reasoning: if there is a retransmission due to timeout, then there is significant loss in the network, and senders should back \*way\* off.

## 9. Reflection

- TCP has been a massive success, requires no changes to the Internet's infrastructure, is something endpoints can opt-in to, allows the network to be shared among tons of different users, all with different -- and changing -- types of traffic, in a distributed manner.
- BUT: TCP doesn't react to congestion until it's already happening. Is there something better we could do?