CS 640 Introduction to Computer Networks

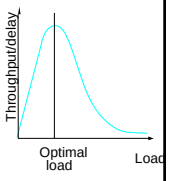
Lecture 26

Today's lecture

- Congestion in networks
- TCP congestion control
  - Additive Increase Multiplicative Decrease
  - Slow start
  - Fast retransmit and fast recovery

Congestion in the Internet

- Checksums are effective for detecting bit errors but they are not the only problem…
- We know that traffic is bursty
  - Statistical multiplexing of ON/OFF sources
  - Heavy-tailed file sizes
  - Routers have limited buffer capacity
  - Packets dropped when buffers full
    - Buffers do protect from short bursts
- Congestion lengthens delays and lowers throughput
  - Standard throughput/load curve



How can we deal with congestion?

- Over-provision networks
  - Very expensive
  - Commonly done
    - Networks designed to normally operate at 5-50% capacity
- Call admission control (phone networks)
- Develop protocols to respond to congestion
  - Route away from congestion
    - Good idea – how can we do it?
  - Retransmit in the face of loss
    - This is the state of the art

Congestion Control Basics

- UDP will send packets at any specified rate
  - Does not have mechanisms to handle congestion
- Issues:
  - Detecting congestion
  - Reacting to congestion
  - Avoiding congestion
    - Shaping traffic
    - QoS mechanisms
- Transport protocol will deal with congestion…

Congestion control in the Internet

- TCP implements congestion control
  - Detects congestion through packet losses
  - Reduces rate aggressively in response to congestion
  - Increases rate cautiously to use up available bandwidth
  - Works well for large flows
- Why the Internet doesn't experience congestion collapse
  - Backbones overprovisioned
  - TCP congestion control
  - Sources' rate limited by nearest bottleneck link

Today's lecture

- Congestion in networks
- TCP congestion control
  - Additive Increase Multiplicative Decrease
  - Slow start
  - Fast retransmit and fast recovery

TCP RENO Overview

- Standard TCP functions in last lecture
  - Connections, reliability, RTT calculation, etc.
- Congestion control/management
  - Additive Increase/ Multiplicative Decrease (AIMD)
  - Fast Retransmit/Fast Recovery
  - Slow Start

TCP Congestion Control

- Idea
  - Assumes best-effort network (FIFO or FQ routers) each source determines network capacity for itself
  - Uses implicit feedback
  - ACKs pace transmission (*self-clocking*)
- Challenge
  - Determining the available capacity in the first place
  - Adjusting to changes in the available capacity

Additive Increase/Multiplicative Decrease

- Objective: adjust to changes in the available capacity
- New state variable per connection: **`CongestionWindow`**
  - limits how much data source has in transit

```
MaxWin = MIN(CongestionWindow,
             AdvertisedWindow)
EffWin = MaxWin - (LastByteSent -
                   LastByteAcked)
```

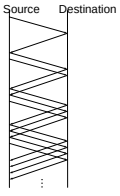
- Idea:
  - increase **`CongestionWindow`** when congestion goes down
  - decrease **`CongestionWindow`** when congestion goes up

AIMD (cont)

- Question: how does the source determine whether or not the network is congested?
- Answer: a timeout occurs
  - timeout signals that a packet was lost
  - packets are seldom lost due to transmission error
  - lost packet implies congestion
  - RTO calculation is critical

AIMD (cont)

- Algorithm
  - increment **`CongestionWindow`** by one packet per RTT (*linear increase*)
  - divide **`CongestionWindow`** by two on timeouts (*multiplicative decrease – fast!!*)
  - **`CongestionWindow`** always >= 1 MSS



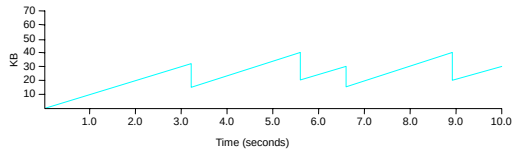
- In practice: increment a little for each ACK

```
Increment = 1/CongestionWindow
CongestionWindow += Increment
```

MSS = max segment size = size of a single packet

AIMD (cont)

- Trace: sawtooth behavior

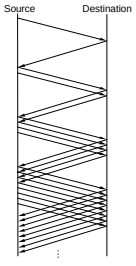


Today's lecture

- Congestion in networks
- TCP congestion control
  - Additive Increase Multiplicative Decrease
  - Slow start
  - Fast retransmit and fast recovery

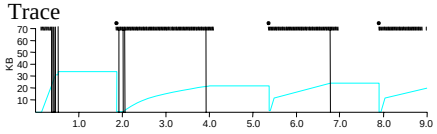
Slow Start

- Objective: determine the available capacity when connection opened
  - Additive increase is too slow
    - One additional packet per RTT
- Idea:
  - begin with `CongestionWindow` = 1 pkt
  - double `CongestionWindow` each RTT (increment by 1 packet for each ACK)
  - This is exponential increase to probe for available bandwidth
- SSTHRESH indicates when to begin additive increase



Slow Start contd.

- Exponential growth, but slower than all at once
- Used…
  - when first starting connection
  - when connection goes dead waiting for timeout
- Trace

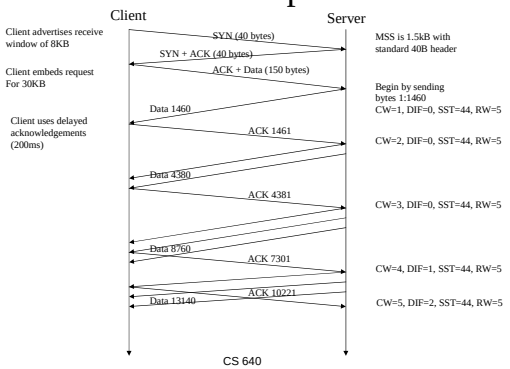


- Problem: lose up to half a **`CongestionWindow`**'s worth of data

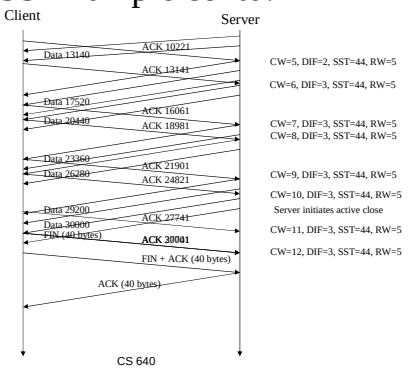
SSTHRESH and CWND

- SSTHRESH called `CongestionThreshold` in book
- Typically set to very large value on connection setup
- Set to one half of `CongestionWindow` on packet loss
  - So, SSTHRESH goes through multiplicative decrease for each packet loss
  - If loss is indicated by timeout, set `CongestionWindow = 1`
    - SSTHRESH and `CongestionWindow` always >= 1 MSS
- After loss, when new data is ACKed, increase CWND
  - Manner depends on whether we're in slow start or congestion avoidance

SS Example



SS Example contd.

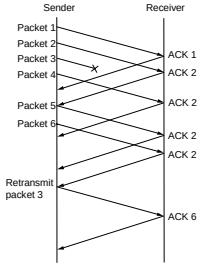


Today's lecture

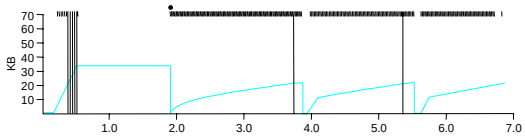
- Congestion in networks
- TCP congestion control
  - Additive Increase Multiplicative Decrease
  - Slow start
  - Fast retransmit and fast recovery

Fast Retransmit and Fast Recovery

- Problem: coarse TCP timeouts lead to idle periods
- Fast retransmit: use 3 duplicate ACKs to trigger retransmission
- Fast recovery: start at SSTHRESH and do additive increase after fast retransmit



Fast Retransmit Results



- This is a graph of fast retransmit only
  - Avoids some of the timeout losses
- Fast recovery
  - skip the slow start phase in this graph at 3.8 and 5.5 sec
  - go directly to half the last successful **`CongestionWindow (ssthresh)`**