Lecture 15:
Congestion Control
Overview

- **Yesterday:**
  - TCP & UDP overview
  - Connection setup
  - Flow control: resource exhaustion at end node

- **Today: Congestion control**
  - Resource exhaustion within the network
Today

- How fast should a sending host transmit data?
  - Not too fast, not too slow, just right…
- Should not be faster than the receiver can process
  - Flow control (last class)
- Should not be faster than the sender’s share
  - Bandwidth allocation
- Should not be faster than the network can process
  - Congestion control
- Congestion control & bandwidth allocation are separate ideas, but frequently combined
Resource Allocation

- How much bandwidth should each flow from a source to a destination receive when they compete for resources?
- What is a “fair” allocation?
Statistical Multiplexing

1. The bigger the buffer, the lower the packet loss.
2. If the buffer never goes empty, the outgoing line is busy 100% of the time.
Buffer intended to absorb bursts when input rate > output
But if sending rate is persistently > drain rate, queue builds
Dropped packets represent wasted work; goodput < throughput (goodput = useful throughput)
Drop-Tail Queuing

Network Load

Goodput

Latency

Congestive packet loss

Congestion collapse

Network Load
Congestion Collapse

- Rough definition: “When an increase in network load produces a decrease in useful work”

- Why does it happen?
  - Sender sends faster than **bottleneck link** speed
    » What's a bottleneck link?
  - Packets queue until dropped
  - In response to packets being dropped, sender retransmits
    » Retransmissions further congest link
  - All hosts repeat in steady state…
Mitigation Options

- **Increase network resources**
  - More buffers for queuing
  - Increase link speed
  - Pros/Cons of these approaches?

- **Reduce network load**
  - Send data more slowly
  - How much more slowly?
  - When to slow down?
Designing a Control

- Open loop
  - Explicitly reserve bandwidth in the network in advance of sending (next class, a bit)

- Closed loop
  - Respond to feedback and adjust bandwidth allocation

- Network-based
  - Network implements and enforces bandwidth allocation (next class)

- Host-based
  - Hosts are responsible for controlling their sending rate to be no more than their share

- What is typically used on the Internet? Why?
Proactive vs. Reactive

- **Congestion avoidance**: try to stay to the left of the “knee”
- **Congestion control**: try to stay to the left of the “cliff”
Challenges to Address

- How to detect congestion?
- How to limit sending data rate?
- How fast to send?
Detecting Congestion

- **Explicit congestion signaling**
  - Source Quench: ICMP message from router to host
    » Problems?
  - DECBit / Explicit Congestion Notification (ECN):
    » Router *marks* packet based on queue occupancy (i.e. indication that packet encountered congestion along the way)
    » Receiver tells sender if queues are getting too full (typically in ACK)

- **Implicit congestion signaling**
  - Packet loss
    » Assume congestion is primary source of packet loss
    » Lost packets (timeout, NAK) indicate congestion
  - Packet delay
    » Round-trip time increases as packets queue
    » Packet inter-arrival time is a function of bottleneck link
Throttling Options

- **Window-based** *(TCP)*
  - Artificially constrain number of outstanding packets allowed in network
  - Increase window to send faster; decrease to send slower
  - **Pro**: Cheap to implement, good failure properties
  - **Con**: Creates traffic *bursts* (requires bigger buffers)

- **Rate-based** *(many streaming media protocols)*
  - Two parameters (period, packets)
  - Allow sending of $x$ packets in period $y$
  - **Pro**: smooth traffic
  - **Con**: fine-grained per-connection timers, what if receiver fails?
Choosing a Send Rate

- Ideally: Keep equilibrium at “knee” of power curve
  - Find “knee” somehow
  - Keep number of packets “in flight” the same
    - E.g., don’t send a new packet into the network until you know one has left (i.e. by receiving an ACK)
  - What if you guess wrong, or if bandwidth availability changes?

- Compromise: adaptive approximation
  - If congestion signaled, reduce sending rate by x
  - If data delivered successfully, increase sending rate by y
  - How to relate x and y? Most choices don’t converge…
TCP’s Probing Approach

- Each source independently probes the network to determine how much bandwidth is available
  - Changes over time, since everyone does this
- Assume that packet loss implies congestion
  - Since errors are rare; also, requires no support from routers
Basic TCP Algorithm

- Window-based congestion control
  - Allows congestion control and flow control mechanisms to be unified
  - $r\text{win}$: advertised flow control window from receiver
  - $c\text{wnd}$: congestion control window estimated at sender
    » Estimate of how much outstanding data network can deliver in a round-trip time
  - Sender can only send $\text{MIN}(r\text{win},c\text{wnd})$ at any time
- Idea: decrease $c\text{wnd}$ when congestion is encountered; increase $c\text{wnd}$ otherwise
- Question: how much to adjust?
Congestion Avoidance

- Goal: Adapt to changes in available bandwidth

- Additive increase, Multiplicative Decrease (AIMD)
  - Increase sending rate by a constant (e.g. by 1500 bytes)
  - Decrease sending rate by a linear factor (e.g. divide by 2)

- Rough intuition for why this works
  - Let $L_i$ be queue length at time $i$
  - In steady state: $L_i = N$, where $N$ is a constant
  - During congestion, $L_i = N + yL_{i-1}$, where $y > 0$
  - Consequence: queue size increases multiplicatively
    » Must reduce sending rate multiplicatively as well
AIMD

- Increase slowly while we believe there is bandwidth
  - Additive increase per RTT
  - $Cwnd += 1$ full packet / RTT

- Decrease quickly when there is loss (went too far!)
  - Multiplicative decrease
  - $Cwnd /= 2$
TCP Congestion Control

Only $W$ packets may be outstanding

Rule for adjusting congestion window ($W$)
- If an ACK is received: $W \leftarrow W + \frac{1}{W}$
- If a packet is lost: $W \leftarrow \frac{W}{2}$
Goal: quickly find the equilibrium sending rate

- Quickly increase sending rate until congestion detected
- Remember last rate that worked and don’t overshoot it

Algorithm:
- On new connection, or after timeout, set $cwnd=1$ full pkt
- For each segment acknowledged, increment $cwnd$ by 1 pkt
- If timeout then divide $cwnd$ by 2, and set $ssthresh = cwnd$
- If $cwnd >= ssthresh$ then exit slow start

- Why called slow? Its exponential after all…
Slow Start Example

Sender

cwnd=1

1

Ack 2

2

3

Ack 3

Ack 4

4

5

6

7

Ack 5

Ack 6

Ack 7

Ack 8

Receiver

round-trip times

cwnd

0 1 2 3 4 5 6 7 8

0 50 100 150 200 250 300

22
Basic Mechanisms

Slow Start + Congestion Avoidance

- Timeout
- Congestion avoidance
- ssthresh

Factors:
- cwnd
- round-trip times
- Slow start

Conclusion:
- Basic mechanisms for network flow control
Fast Retransmit & Recovery

- Fast retransmit
  - Timeouts are slow (1 second is fastest timeout on many TCPs)
  - When packet is lost, receiver still ACKs last in-order packet
  - Use 3 duplicate ACKs to indicate a loss; detect losses quickly
    » Why 3? When wouldn’t this work?

- Fast recovery
  - Goal: avoid stalling after loss
  - If there are still ACKs coming in, then no need for slow start
  - If a packet has made it through -> we can send another one
  - Divide cwnd by 2 after fast retransmit
  - Increment cwnd by 1 full pkt for each additional duplicate ACK
Fast Retransmit Example

Sender

1
Ack 2
2
3
Ack 3
Ack 4
4
5
Ack 4
Ack 4
Ack 4
Ack 4
4

3 Dup Acks

Fast recovery (increase cwnd by 1)

Receiver

Fast retransmit
More Sophistication

Slow Start + Congestion Avoidance + Fast Retransmit + Fast Recovery

round-trip times

cwnd

Fast recovery
Delayed ACKs

- In request/response programs, want to combine an ACK to a request with a response in the same packet
  - Delayed ACK algorithm:
    - Wait 200ms before ACKing
    - Must ACK every other packet (or packet burst)
  - Impact on slow start?

- Must not delay **duplicate ACKs**
  - Why? What is the interaction with the congestion control algorithms?
Short Connections

- Short connection: only contains a handful of pkts
- How do short connections and Slow-Start interact?
  - What happens when a packet is lost during Slow-Start?
  - Will short connections be able to use full bandwidth?

- Do you think most flows are short or long?
- Do you think most traffic is in short flows or long flows?
Open Issues

- TCP is designed around the premise of cooperation
  - What happens to TCP if it competes with a UDP flow?
  - What if we divide cwnd by 3 instead of 2 after a loss?

- There are a bunch of magic numbers
  - Decrease by 2x, increase by 1/cwnd, 3 duplicate acks, initial timeout = 3 seconds, etc

- But overall it works quite well!
TCP actively probes the network for bandwidth, assuming that a packet loss signals congestion.

The congestion window is managed with an additive increase/multiplicative decrease policy. It took fast retransmit and fast recovery to get there.

Slow start is used to avoid lengthy initial delays. Ramp up to near target rate, then switch to AIMD.

Fast recovery is used to keep network “full” while recovering from a loss.
For next time…

- Last class: Resource management in the network (i.e., Quality of Service)
- P&D 6.1-6.2, 6.5.1