Congestion Control

Kai Shen

Principles of Congestion Control

Congestion:
- informally: “too many sources sending too much data too fast for the network to handle”

Results of congestion:
- long delays (e.g. queueing in router buffers)
- lost packets (e.g. buffer overflow at routers)

Congestion Control

Congestion threatens the growth of the Internet
- tragedy of commons

Two broad approaches towards congestion control:

Network-assisted congestion control:
- routers participate in congestion control
- routers provide feedback to end systems that slow down when necessary

End-end congestion control:
- no explicit help/information from network
- congestion inferred from end-system observed loss, delay

Network-assisted Congestion Control: ATM

- RM (resource management) cells:
  - sent by sender, interspersed with data cells
  - NI bit: no increase in rate (mild congestion)
  - CI bit: congestion indication
  - ER (explicit rate) field: congested switch may lower ER value in cell

- RM cells returned to sender by receiver, with bits intact
  - sender control its rate based on information received in RM cells
TCP Congestion Control

- **Mechanism:** sender controls the sending rate by adjusting the number of packets allowed in flight simultaneously (size of the sliding window)
- **Objective:** figure out the highest rate that does not cause network congestion

How does sender perceive congestion?

- Roughly: \( \text{throughput} = \frac{\text{CongWin}}{\text{RTT}} \)
- Dynamically control CongWin, based on perceived network congestion
- **Goal:** maintain a high rate that does not cause congestion

- **CongWin** is maximum allowed in-flight or un-ACKed bytes
- Stop exponential growth: when there is a loss event or when CongWin reaches a Threshold.
- **Threshold** divides between two states:
  - when you are far away from causing congestion, so you do not need to worry about congestion
  - when you are very close to causing congestion, do need to be very careful
- **Threshold** is maintained based on past history:
  - set to half of CongWin at a loss event
  - initially set to a large value so it has no effect

**“Slow” Start Phase**

- At the beginning, CongWin = 1 MSS
- Available bandwidth may be >> MSS/RTT
  - Example: MSS = 500 bytes & RTT = 200 msec
    - initial rate = 20 kbps
    - desirable to quickly ramp up to respectable rate
- **Exponential increase:** double CongWin after every RTT in the absence of loss events
- In this phase, the CongWin starts at a slow rate, but it grows fast
- When should we stop exponential growth?

**When does Slow Start become Steady State?**

- **Threshold** is maintained based on past history:
  - set to half of CongWin at a loss event
  - initially set to a large value so it has no effect
Steady State Phase: AIMD

Additive increase:
- Increase CongWin by 1 MSS every RTT in the absence of loss events.

Multiplicative decrease:
- Cut CongWin in half after 3 duplicate ACKs.

Handling Loss Events

- After 3 dup ACKs:
  - CongWin is cut in half, then grows linearly.
- But after timeout event:
  - CongWin instead set to 1 MSS, fall back to slow start, the very beginning of the connection.

Philosophy:
- Timeout before 3 dup ACKs is more alarming because 3 dup ACKs indicates network capable of delivering some segments.

Summary

- At start phase, window starts at 1 MSS and grows exponentially.
- When CongWin is above threshold, sender is in steady state phase, window grows linearly.
- When a triple duplicate ACK occurs, CongWin is cut in half.
- When timeout occurs, threshold set to CongWin/2 and CongWin is set to 1 MSS (start again).

Congestion Control in Action
Congestion Control Algorithms

- Reno is what we described.
- Tahoe (a predecessor)
  - does not consider three duplicate ACKs as a loss event.
- New Reno (an enhancement)
  - Congestion is most related to the number of in-flight segments. But Reno send window limits the number of unacked segments, many of which are not in-flight.
    - send one more segment when receiving a duplicate ACK.
- Vegas
- CUBIC (Linux)
  - More sophisticated slow start, steady state management
  - Window growth based on time, not ACKs

Heuristics-based Approaches

- Many heuristics-based parameters, little understanding on how good it is compared to alternatives, or how far it is from optimal.

Outline

- Principles of congestion control
- Congestion control in TCP
  - Impact of congestion control on fairness
  - Impact of congestion control on TCP efficiency

Fairness of TCP Congestion Control

Fairness goal:
If K TCP sessions share same bottleneck link of bandwidth R, each should have average rate of R/K
AIMD is Fair

Two competing sessions:
- Additive increase gives slope of 45°, as throughputs of both sessions increase at the same speed
- Multiplicative decrease decreases throughput proportionally

Fairness with UDP

- UDP is not congestion-controlled
  - UDP apps can send data as fast as they want, despite losses (its own losses as well as others)
  - Multimedia applications
- TCP is network friendly while UDP is not
- Regulate UDP traffic within the network

TCP Efficiency

Q: How long does it take to receive an object from a Web server after sending a request?
- Data transmission delay
- TCP connection establishment
- Congestion control (slow start)

Case study:
- Assume fixed congestion window: W segments
- More complicated with full TCP congestion control
Fixed Congestion Window (1)

Notation, assumptions:
- Link rate: $R$
- Object size: $O$
- Fixed window size: $WS$
- No loss, no corruption

Case 1: congestion window is large
- ACK for first segment in window returns before window's worth of data sent

$$\text{delay} = 2\text{RTT} + \frac{O}{R}$$

Fixed Congestion Window (2)

Notation, assumptions:
- Link rate: $R$
- Object size: $O$
- Fixed window size: $WS$
- No loss, no corruption

Case 2: congestion window is not large
- wait for ACK after sending window's worth of data sent

$$\text{delay} = 2\text{RTT} + \frac{O}{R} + (K-1)[\frac{S}{R} + \text{RTT} - \frac{WS}{R}]$$

where $K = \frac{O}{WS}$

TCP Efficiency with Slow Start

- Small window size in the slow start process $\Rightarrow$ inefficiency
- Inefficiency is amortized over long-running connections
  - persistent connection in HTTP/1.1

Disclaimer

- Parts of the lecture slides contain original work of James Kurose, Larry Peterson, and Keith Ross. The slides are intended for the sole purpose of instruction of computer networks at the University of Rochester. All copyrighted materials belong to their original owner(s).