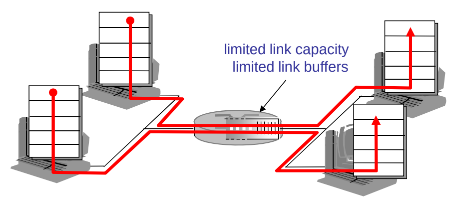
Congestion Control

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:

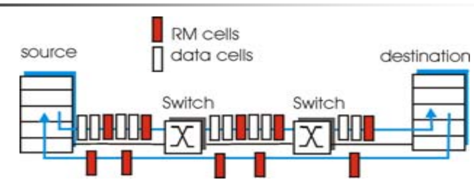
Network-assisted congestion control:

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

End-end congestion control:

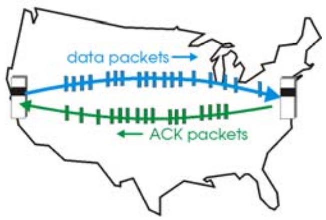
- no explicit help/information from network
- congestion inferred from end-system observed loss, delay

Network-assisted Congestion Control: ATM



- RM (resource management) cells:
  - sent by sender, interspersed with data cells
- bits in RM cell set by switches ("*network-assisted*")
  - 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

TCP Congestion Control

- **CongWin** is maximum allowed in-flight or un-ACKed bytes
- Roughly:
  - throughput = CongWin/RTT
- dynamically control **CongWin**, based on perceived network congestion
- Goal: maintain a high rate that does not cause congestion

How does sender perceive congestion?

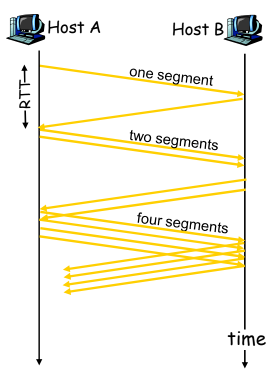
- loss event = timeout *or* triple duplicate ACKs
- timeout before 3 dup ACKs is more alarming
  - 3 dup ACKs indicates network capable of delivering some segments

two phases for TCP congestion control:

- start phase (having little idea of where the good rate is)
- steady state phase (being quite close to the good rate)

"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?



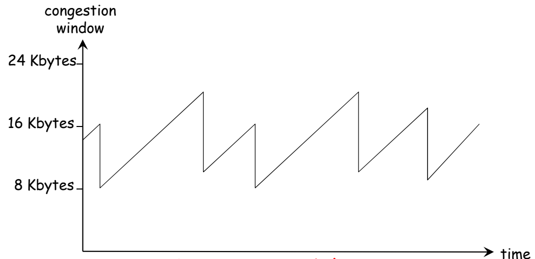
When does Slow Start become Steady State?

- 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

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



Long-term stability

Handling Loss Events

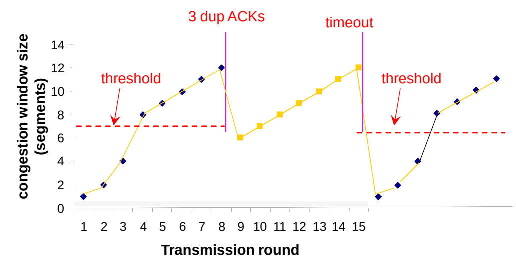
- After 3 dup ACKs:
  - **CongWin** is cut in half, it then grows linearly
- But after timeout event:
  - **CongWin** instead set to 1 MSS, fall back to slow start - the very beginning of the connection

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

10/19/2011 CSC 257/457 - Fall 2011 13

Heuristics-based Approaches

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

10/19/2011 CSC 257/457 - Fall 2011 14

Outline

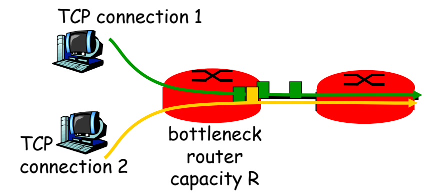
- Principles of congestion control
- Congestion control in TCP
- Impact of congestion control on fairness
- Impact of congestion control on TCP efficiency

10/19/2011 CSC 257/457 - Fall 2011 15

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

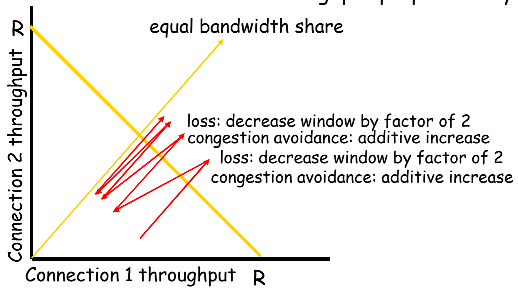


10/19/2011 CSC 257/457 - Fall 2011 16

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



10/19/2011 CSC 257/457 - Fall 2011 17

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

10/19/2011 CSC 257/457 - Fall 2011 18

Fairness with Parallel TCP Connections

- Nothing prevents an app from opening parallel connections between two hosts.
- Example: link of rate R currently supporting 9 connections.
  - new app uses 1 TCP connection, gets rate R/10
  - new app asks for 9 TCP connections, gets R/2 !

10/19/2011 CSC 257/457 - Fall 2011 19

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)

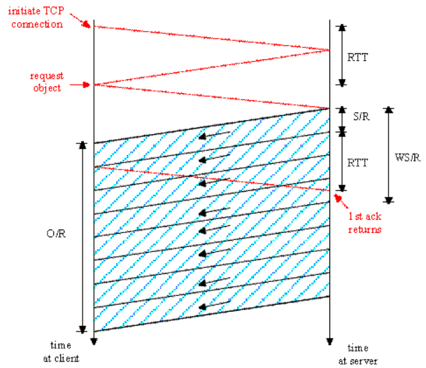
Case study:

- assume fixed congestion window: W segments
- more complicated with full TCP congestion control

10/19/2011 CSC 257/457 - Fall 2011 20

Fixed Congestion Window (1)

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



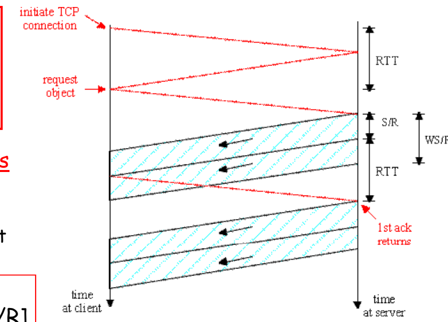
Case 1: congestion window is large

- ACK for first segment in window returns before window's worth of data sent

delay = 2RTT + 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

delay = 2RTT + O/R
+ (K-1)[S/R + RTT - WS/R]
where K = O/WS

TCP Efficiency with Slow Start

- Small window size in the slow start process ⇒ 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).