Congestion Control

Daniel Zappala

CS 460 Computer Networking
Brigham Young University

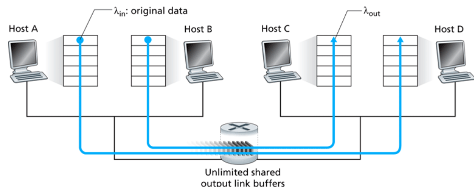
Congestion Control

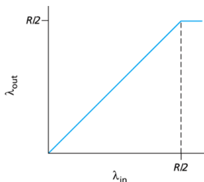
- how do you send as fast as possible, without overwhelming the network?
- challenges
  - the fastest speed you can send changes all the time, due to many connections stopping and starting all the time
  - you must have a stable solution
  - you must provide fairness among different connections

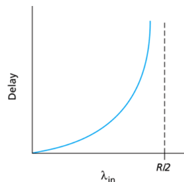
Definition

- congestion: hosts sending data at a rate that exceeds network capacity
- effects
  - delay
  - packet loss

Infinite Buffering

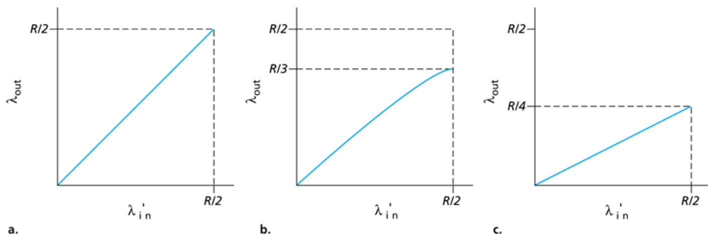






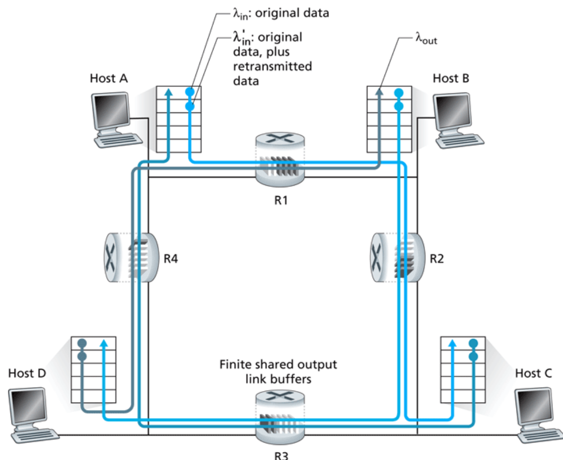
- infinite buffering, no retransmission
- exponential arrival rate, characterized by an average
  - **median** $<$ **average**
- delay grows as arrival rate grows
- each link (and path) has a maximum achievable throughput

Finite Buffering



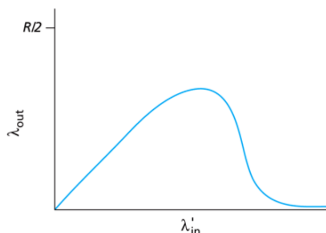
- finite buffering, retransmission
- λ'_{in} = original rate plus retransmissions
- (a) perfect sender, no loss, $\lambda_{in} = \lambda'_{in} = \lambda_{out}$
- (b) perfect retransmission, $\lambda'_{in} > \lambda_{out}$
  - illustrates cost of congestion - for equal amount of work, less throughput
- (c) imperfect retransmission
  - illustrates cost of unneeded retransmissions

Four Senders



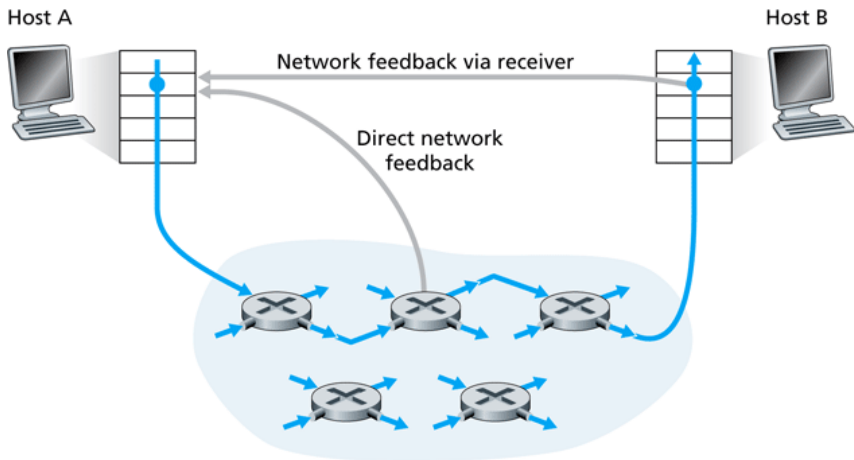
- finite buffering
- multihop paths
- timeouts and retransmissions
- what happens as λ'_{in} increases?

Congestion Collapse



- congestion collapse: as offered load increases, throughput suddenly drops to zero
- **when a packet is dropped, the capacity used at early routers is wasted**
- October 1986: this happened on the Internet!
- Van Jacobson and Michael J. Karels, Congestion Avoidance and Control. ACM SIGCOMM, 1988.

Approaches to Congestion Control



Approaches to Congestion Control

- network-assisted congestion control
  - routers provide feedback to hosts
  - one bit
    - set a single bit to indicate congestion on forward path
    - bit is mirrored on ACKs sent back to sender
    - used by DECbit (early suggestion for TCP), ATM (telephony protocols for integrated audio and video)
    - now used by TCP/ECN (latest proposal/implementation of TCP to react to congestion)
    - www.icir.org/floyd/ecn.html
  - explicit rate
    - tell sender explicit rate it should send at
- end-to-end congestion control
  - no explicit feedback from network
  - congestion inferred from observing loss at the sender
  - used by TCP

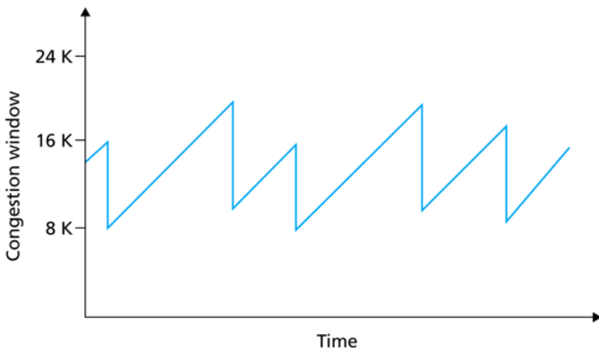
TCP Congestion Control

TCP Congestion Control

- end-to-end (no network assistance)
- window-based
    - sender's window size limits the amount of outstanding packets
    - conservation of packets: never send a new packet until an old packet leaves
    - $LastByteSent - LastByteAcked \leq CongWin$
- can compute a rate from the window size
    - $rate = \frac{CongWin}{RTT} bytes/s$
- congestion window must be dynamic, reacting to network congestion
    - loss event: timeout or 3 duplicate acks (Fast Retransmit)
    - reduce rate by decreasing window after each loss
- three important TCP mechanisms
    - AIMD
    - slow start
    - conservative rate after timeout

AIMD – Congestion Avoidance

- additive increase
  - increase CongWin by 1 MSS every RTT with no loss
- multiplicative decrease
  - decrease CongWin by 1/2 after every loss event
- results in a familiar sawtooth pattern



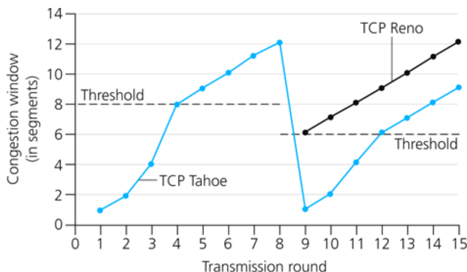
Slow Start

- when connection starts, set CongWin = 1 MSS
  - example: MSS = 500 bytes, RTT = 2 ms
  - initial rate = 20 kbps
- available bandwidth likely >> MSS/RTT
  - want to quickly increase rate to take advantage of available bandwidth
- **increase rate exponentially until first loss event**
  - double CongWin every RTT
  - done by incrementing CongWin by 1 MSS for every ACK
  - example: 1, 2, 4, 8, 16 ...

Conservative Rate after Timeout (TCP Reno)

- after 3 duplicate ACKs
  - decrease CongWin by 1/2
  - begin AIMD (linear increase)
  - called Fast Recovery
- after timeout event
  - set CongWin to 1 MSS
  - begin slow start (exponential increase)
  - when window reaches a threshold (1/2 of CongWin when loss event occurred), use AIMD
- philosophy
  - 3 duplicate ACKs indicates an isolated loss event – network is still delivering the rest of the data
  - timeout event indicates severe loss event

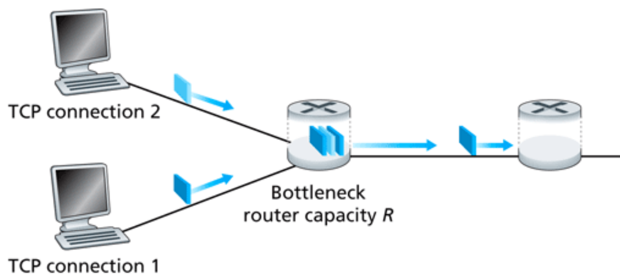
Visualizing TCP Congestion Control



- periods of slow start, AIMD
- Threshold = 1/2 CongWin when loss occurs
- TCP Tahoe: set CongWin to 1 on every loss, use slow start for 1 to Threshold
- TCP Reno: Fast Recovery - keep sending a new packet for each duplicate ack (beyond 3) until new ack received

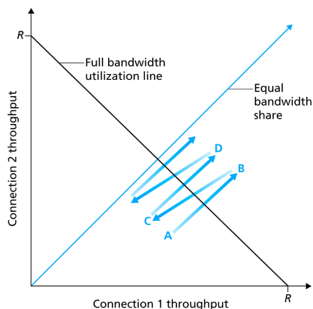
TCP Fairness

Classic Fairness Scenario



- would like each connection over the bottleneck link to get 1/2 the bandwidth
- with N connections, want each to get 1/N of the link capacity

TCP Fairness Results from AIMD



- A-B: additive increase, slope of 1 as throughput increases
- B-C: multiplicative decrease when loss occurs
- C-D: additive increase again
- converge at fair share
- **requires that both connections play fair**

Unfairness

- fairness only holds if RTT for each connection is equal
- multimedia applications often use UDP
  - like to have a constant rate for voice/video
  - tolerate loss
  - need TCP-friendly rate control for UDP
  - provide congestion control without reliability
  - Datagram Congestion Control Protocol (DCCP)
  - http://www.icir.org/kohler/dcp/
- many applications use multiple parallel connections
  - web browser fetching one image per connection
  - 1 app - 1 TCP connection vs 1 app - 9 TCP connections
  - one user gets 1/10 of bandwidth, the other gets 9/10

TCP Versions

Tahoe and Reno

- Tahoe (1988)
  - slow start
  - AIMD
  - fast retransmit
- Reno (1990)
  - Tahoe + Fast Recovery
- NewReno: RFC 2582 (1995 - 1999)
  - fixes Reno for the case when multiple segments are lost in a row

Vegas

- complete rewrite of TCP (1994)
  - separates reliability from congestion control
  - rate-based instead of window based
- observe the RTT samples
  - minimum RTT - ideal delay if there is no queuing delay
  - if RTT is higher than minimum, then there must be congestion, so reduce the rate
  - if RTT is close to minimum, then there must not be any congestion, so increase the rate
  - attempt to keep a small queue in the router
- requires changes to sender and receiver
- lots of controversy, never widely adopted

SACK

- RFC 2018, 2883 (1996 - 1999)
- optimization: multiple segments lost from one window
- with cumulative ACKs, a TCP sender can only learn about a single lost packet per RTT
  - sender transmits segments 1 - 10
  - sender gets duplicate ACKS for segment 3
  - sender knows 3 is lost, but what else?
- selective acknowledgments (SACK) let a TCP sender know exactly which packets have been received
  - attach list of segments received out of order
  - ACK 3, SACK 4:5,7,9:10
- TCP sender can now retransmit the missing packets immediately, subject to congestion control constraints
- turned on by default in Windows 98, see /proc/sys/net/ipv4 on Linux

ECN

- RFC 3168 (1994 - 2000)
- explicit congestion notification
  - routers set a bit in the IP header to indicate that persistent congestion is occurring (queue length is over a threshold for a period of time)
  - receiver sets a bit in the TCP header when sending an ACK
- TCP sender slows down as if loss event occurred

High Speed, Long Distance

- very high speed (Gbps) networks require a very large window to fully utilize bandwidth (window = number of outstanding packets)
- AIMD increases the window too slowly and decreases the window too aggressively when the window size is very large
- a number of variants address this problem
  - BIC (default in newer Linux kernels) (2004)
  - CUBIC (improvement of BIC) (2006)
  - FAST (2002)