

Computer Networks

Congestion Control

Based on Computer Networking, 4th Edition by Kurose and Ross

Stan Kurkovsky

Principles of Congestion Control

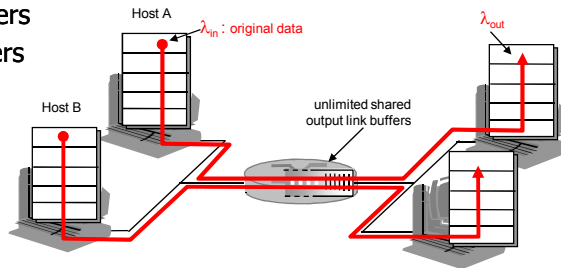
Congestion:

- informally: “too many sources sending too much data too fast for *network* to handle”
- different from flow control!
- manifestations:
 - lost packets (buffer overflow at routers)
 - long delays (queueing in router buffers)
- a top-10 problem!

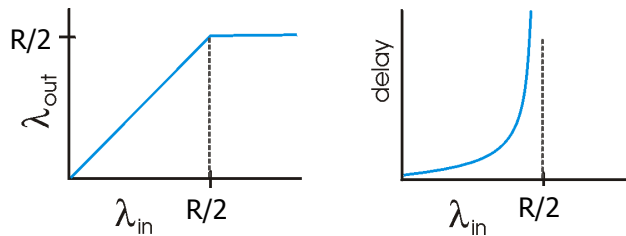
Stan Kurkovsky

Causes/costs of congestion: scenario 1

- two senders, two receivers
- one router, infinite buffers
- no retransmission



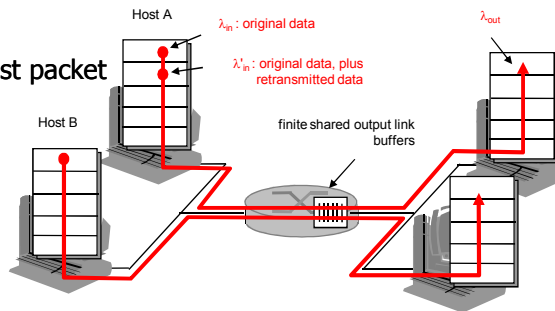
- large delays when congested
- maximum achievable throughput



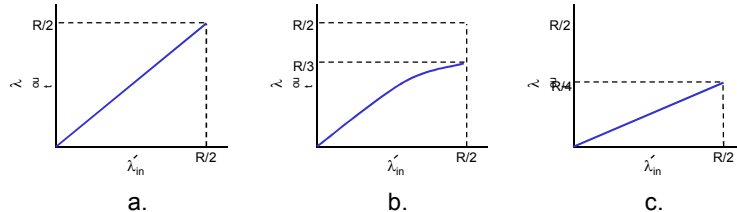
Stan Kurkovsky

Causes/costs of congestion: scenario 2

- one router, *finite* buffers
- sender retransmission of lost packet



- always: $\lambda_{in} = \lambda_{out}$ (goodput)
- "perfect" retransmission only when loss: $\lambda'_{in} > \lambda_{out}$
- retransmission of delayed (not lost) packet makes λ'_{in} larger (than perfect case) for same λ_{out}



"costs" of congestion:

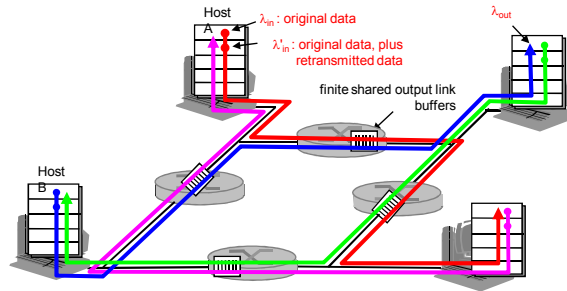
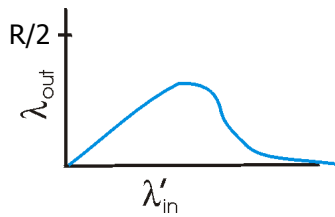
- more work (retrans) for given "goodput"
- unneeded retransmissions: link carries multiple copies of pkt

Stan Kurkovsky

Causes/costs of congestion: scenario 3

- four senders
- multihop paths
- timeout/retransmit

Q: what happens as λ_{in} and λ'_{in} increase ?



Another "cost" of congestion:

- when packet dropped, any "upstream transmission capacity used for that packet was wasted!

Stan Kurkovsky

Approaches towards congestion control

- Two broad approaches towards congestion control

End-end congestion control:

- no explicit feedback from network
- congestion inferred from end-system observed loss, delay
- approach taken by TCP

Network-assisted congestion control:

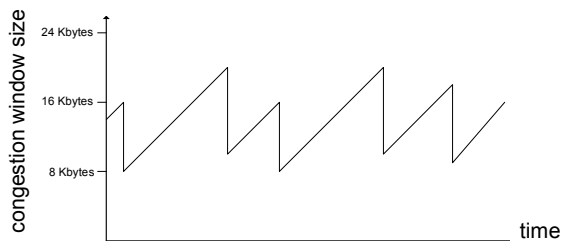
- routers provide feedback to end systems
 - single bit indicating congestion (SNA, DECbit, TCP/IP ECN, ATM)
 - explicit rate sender should send at

Stan Kurkovsky

TCP congestion control: AIMD

- AIMD: additive increase, multiplicative decrease
- **Approach:** increase transmission rate (window size), probing for usable bandwidth, until loss occurs
 - **additive increase:** increase **CongWin** by 1 MSS (max segment size) every RTT until loss detected
 - **multiplicative decrease:** cut **CongWin** in half after loss

Saw tooth
behavior: probing
for bandwidth



Stan Kurkovsky

TCP Congestion Control: details

- sender limits transmission:
 $\text{LastByteSent} - \text{LastByteAcked} \leq \text{CongWin}$
- Roughly,

$$\text{rate} = \frac{\text{CongWin}}{\text{RTT}} \text{ Bytes/sec}$$

- **CongWin** is dynamic, function of perceived network congestion

How does sender perceive congestion?

- loss event = timeout *or* 3 duplicate acks
- TCP sender reduces rate (**CongWin**) after loss event

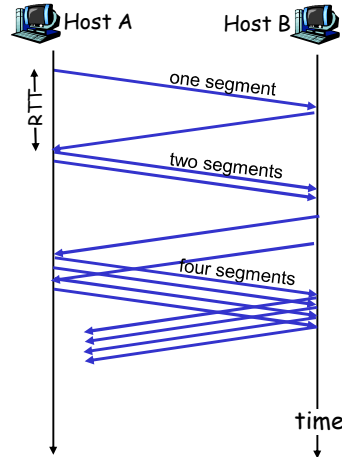
three mechanisms:

- AIMD
- slow start
- conservative after timeout events

Stan Kurkovsky

TCP Slow Start

- When connection begins, **CongWin** = 1 MSS
 - Example: MSS = 500 bytes & RTT = 200 msec
 - initial rate = 20 kbps
- available bandwidth may be \gg MSS/RTT
 - desirable to quickly ramp up to respectable rate
- When connection begins, increase rate exponentially until first loss event:
 - double **CongWin** every RTT
 - done by incrementing **CongWin** for every ACK received
- **Summary:** initial rate is slow but ramps up exponentially fast



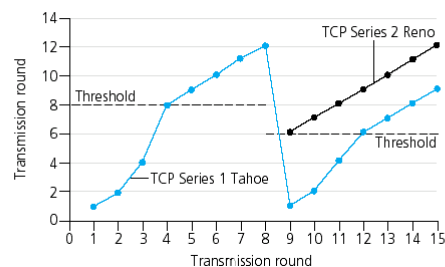
Stan Kurkovsky

Refinement

- Q:** When should the exponential increase switch to linear?
A: When **CongWin** gets to 1/2 of its value before timeout.

Implementation:

- Variable Threshold
- At loss event, Threshold is set to 1/2 of CongWin just before loss event



Stan Kurkovsky

Refinement: inferring loss

- After 3 dup ACKs:
 - **CongWin** is cut in half
 - window then grows linearly
- But after timeout event:
 - **CongWin** instead set to 1 MSS;
 - window then grows exponentially
 - to a threshold, then grows linearly
- 3 dup ACKs indicates network capable of delivering some segments
- timeout indicates a "more alarming" congestion scenario

Stan Kurkovsky

Summary: TCP Congestion Control

- When **CongWin** is below **Threshold**, sender in **slow-start** phase, window grows exponentially.
- When **CongWin** is above **Threshold**, sender is in **congestion-avoidance** phase, window grows linearly.
- When a **triple duplicate ACK** occurs, **Threshold** set to **CongWin/2** and **CongWin** set to **Threshold**.
- When **timeout** occurs, **Threshold** set to **CongWin/2** and **CongWin** is set to 1 MSS.

Stan Kurkovsky

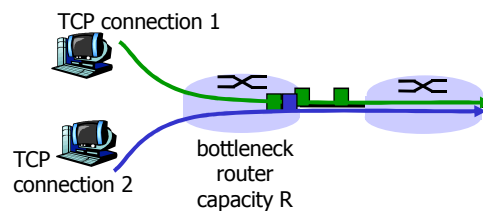
TCP throughput

- What's the average throughput of TCP as a function of window size and RTT?
 - Ignore slow start
- Let W be the window size when loss occurs.
- When window is W , throughput is W/RTT
- Just after loss, window drops to $W/2$, throughput to $W/2RTT$.
- Average throughput: $.75 W/RTT$

Stan Kurkovsky

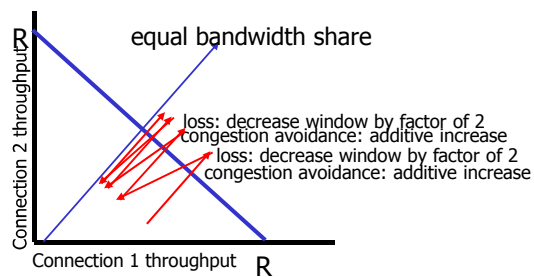
TCP Fairness

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



Two competing sessions:

- Additive increase gives slope of 1, as throughput increases
- multiplicative decrease decreases throughput proportionally



Stan Kurkovsky

TCP Fairness

Fairness and UDP

- Multimedia apps often do not use TCP
 - do not want rate throttled by congestion control
- Instead use UDP:
 - pump audio/video at constant rate, tolerate packet loss
- Research area: TCP friendly

Fairness and parallel TCP connections

- nothing prevents app from opening parallel connections between 2 hosts.
- Web browsers do this
- Example: link of rate R supporting 9 cncctions;
 - new app asks for 1 TCP, gets rate $R/10$
 - new app asks for 11 TCPs, gets $R/2$!

Stan Kurkovsky