CS118 Week 5 Lecture Slides

Chapter 3 outline

- 3.1 Transport-layer services
- 3.2 Multiplexing and demultiplexing
- 3.3 Connectionless transport: UDP
- 3.4 Principles of reliable data transfer
- 3.5 Connection-oriented transport: TCP
  - segment structure
  - reliable data transfer
  - flow control
  - connection management
- 3.6 Principles of congestion control
- 3.7 TCP congestion control

Network Congestion

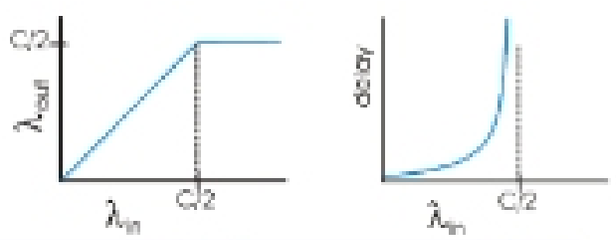
Congestion: "too many sources sending too much data too fast for *network* to handle"

Scenario 1

- 2 senders, 2 receivers
- one router w/ *infinite* buffer
- no retransmission

When congested:

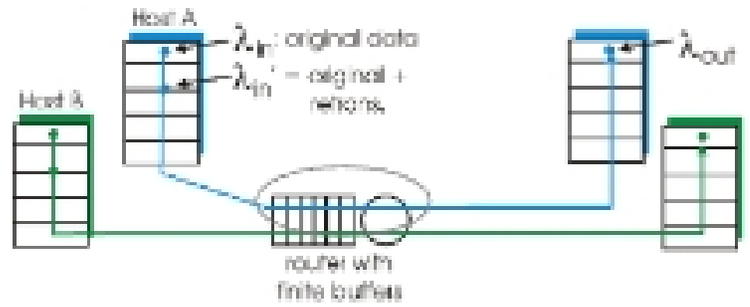
- long delays
- maximum achievable throughput



Congestion: scenario 2

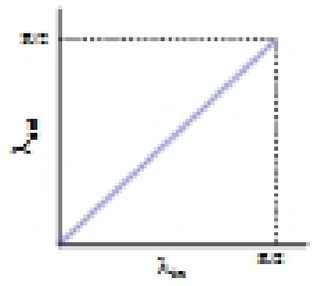
one router, *finite* buffer

senders *retransmit* when timeout



Ideal case:

- Each sender takes turns and sends only when router buffer available

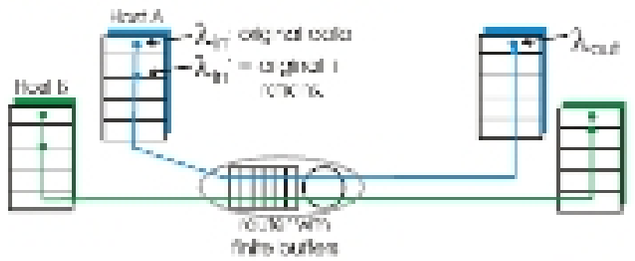


$\lambda_{in} = \lambda_{out}$

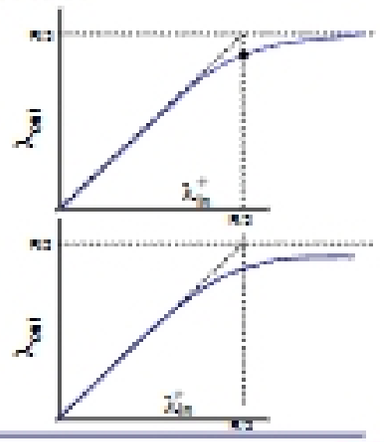
Congestion: scenario 2

one router, *finite* buffer

senders *retransmit* when timeout

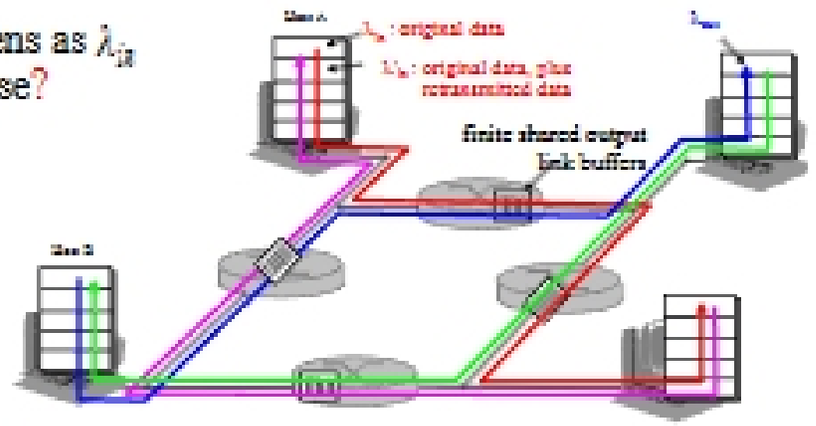


- Packets may get dropped at router due to buffer full
- **Known loss case**: sender only retransmits if packet known to be lost
- **Duplicates**: sender times out prematurely and retransmits, *both* packets are delivered



Congestion scenario 3

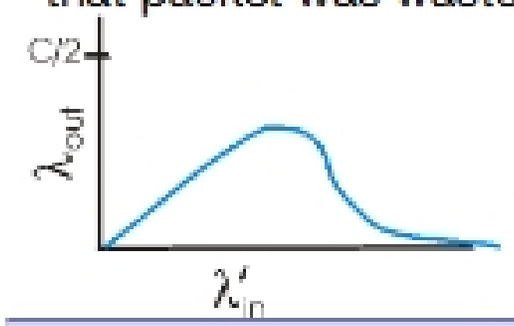
Q: what happens as λ_{in} and λ'_{in} increase?

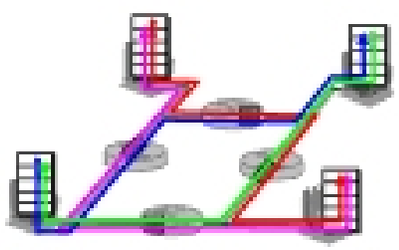


- Long delays
- superfluous retransmissions
- when a packet is dropped, any "upstream transmission capacity" used for that packet was wasted!

Cost of congestion

- unneeded retransmissions: bottleneck link transmitted multiple copies of the same packet, reduce effective throughput
- when a packet dropped further down the road, any "upstream" transmission capacity used for that packet was wasted!





Congestion control: options

Two broad approaches towards congestion control:

Network-assisted congestion control: routers provide feedback to end hosts, such as

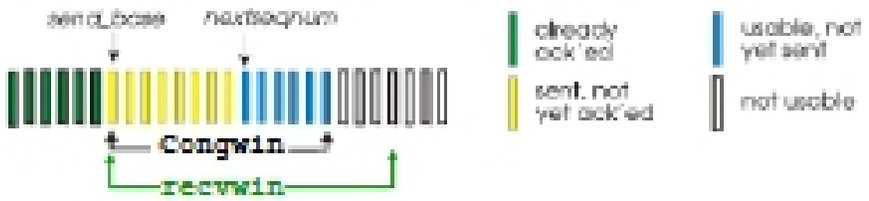
- Single bit congestion indication, or
- Explicit rate that sender should send at

End-end congestion control: no explicit feedback from network

- Hosts infer congestion from observed loss or delay
  - approach taken by TCP

TCP Congestion Control

- Add a "congestion control window" `congwin` on top of flow-control window
- Sender limits: `LastByteSent-LastByteAcked` ≤ `CongWin`



- `CongWin` initialized to 1 packet, increases until congestion
  - How sender infers congestion: packet loss (timeout, or 3 dup. ACKs)
- Upon loss: *decrease* `congwin`, then begin increasing again
  - Two phases: (1)slow start, (2)congestion avoidance
  - `threshold` defines the boundary between the two

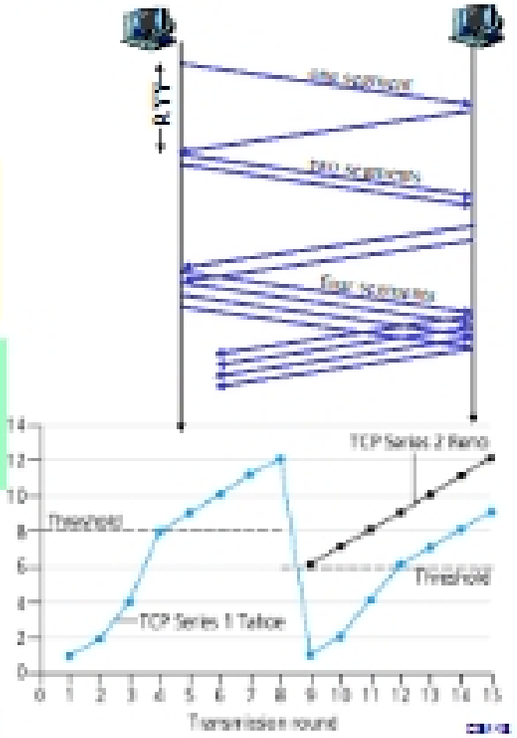
Basic idea: learn from observations

- (mss: maximum segment size)
- when `congwin < threshold`, increase `congwin` exponentially
- when `congwin ≥ threshold`, increase `congwin` linearly
- if packet lost, have gone too far
  - `threshold` = `congwin` / 2
  - If 3 dup. ACKs: network capable of delivering some packets, cut `congwin` to half of its current value
  - If timeout: slow-start again (`congwin` = 1 mss)
- Additive Increase, Multiplicative Decrease (AIMD)

TCP SlowStart & Congestion Avoidance

```
initialize:
   Congwin = 1 mss
   threshold = RcvWindow
if (CongWin < threshold)
{ for every segment ACKed
      Congwin++
} until (loss event)

/* slowstart is over    */
{ for every w segments ACKed:
      Congwin++
} Until (loss event)
/* loss detected */
threshold = Congwin/2
If (3 dup. ACKs)
   Congwin = threshold
Else
   Congwin = 1 mss
```



TCP sender congestion control

State	Event	TCP Sender Action	Commentary
Slow Start (SS)	Received ACK for previously unacked data	CongWin = CongWin + MSS If (CongWin > Threshold) set state to "Congestion Avoidance"	Resulting in a doubling of CongWin every RTT
Congestion Avoidance (CA)	Received ACK for previously unacked data	CongWin = CongWin+MSS * (MSS/CongWin)	Additive increase, resulting in increase of CongWin by 1 MSS every RTT
SS or CA	Loss event detected by 3 duplicate ACK	Threshold = CongWin/2, CongWin = Threshold, Set state to "Congestion Avoidance"	Fast recovery, implementing multiplicative decrease. CongWin will not drop below 1 MSS.
SS or CA	Timeout	Threshold = CongWin/2, CongWin = 1 MSS, Set state to "Slow Start"	Enter slow start
SS or CA	Duplicate ACK	Increment duplicate ACK count for segment being acked	CongWin and Threshold not changed