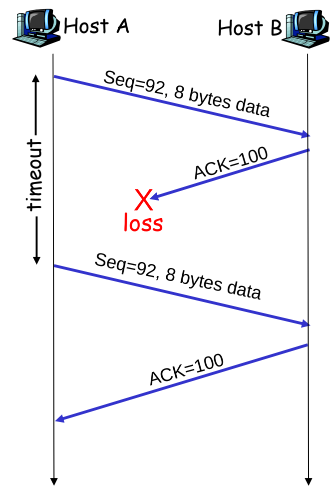
Digital Communication in the Modern World

Transport Layer: TCP timeout optimizations, TCP Flow Control, Congestion

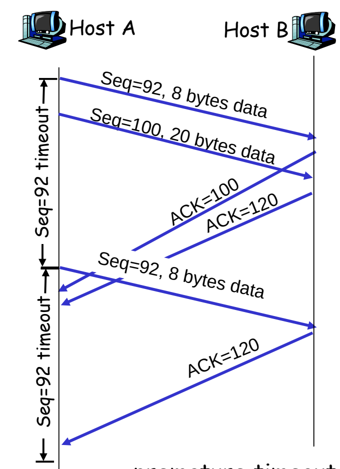
http://www.cs.huji.ac.il/~com1
com1@cs.huji.ac.il

Some of the slides have been borrowed from:
Computer Networking: A Top Down Approach Featuring the Internet,
2nd edition.
Jim Kurose, Keith Ross
Addison-Wesley, July 2002.

TCP: retransmission scenarios

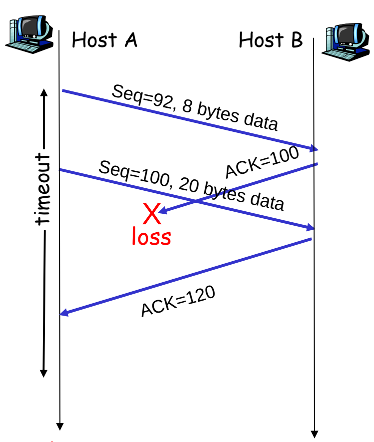


lost ACK scenario



premature timeout

TCP retransmission scenarios (more)



Cumulative ACK scenario

TCP Timeout and Round Trip Time

Q: how to set TCP timeout value?

- longer than RTT
  - but RTT varies
- **too short**: premature timeout
  - unnecessary retransmissions
- **too long**: slow reaction to segment loss

Q: how to estimate RTT?

- `SampleRTT`: measured time from segment transmission until ACK receipt
  - ignore retransmissions
- `SampleRTT` will vary, want estimated RTT "smoother"
  - average several recent measurements, not just current `SampleRTT`

TCP Timeout and Round Trip Time

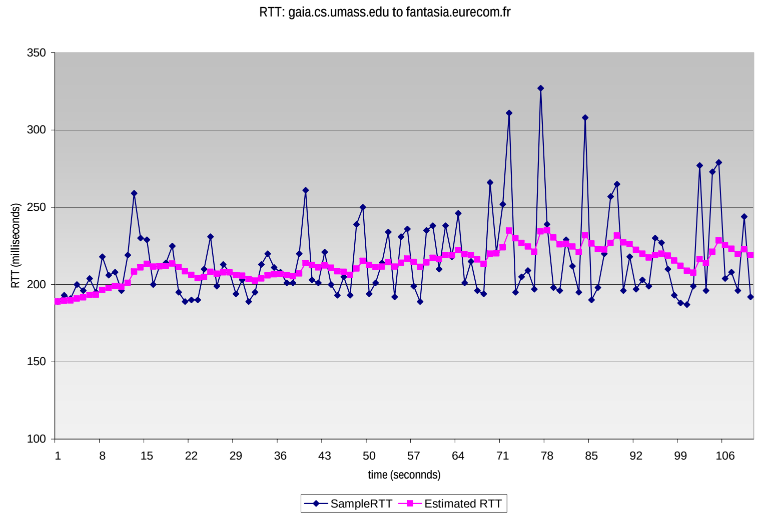
EstimatedRTT = (1-α)*EstimatedRTT + α*SampleRTT

- Exponential Weighted Moving Average (EWMA)
- influence of past sample decreases exponentially fast
- typical value: α = 0.125

EstimatedRTT =

$$\alpha \sum_{j=1}^{n-1} (1-\alpha)^{j} \, SampleRTT_j + (1-\alpha)^n \, SampleRTT_n$$

Example RTT estimation:



TCP Timeout and Round Trip Time

Setting the timeout

- **EstimatedRTT** plus "safety margin"
  - large variation in **EstimatedRTT ->** larger safety margin
- first estimate of how much SampleRTT deviates from EstimatedRTT:

```
DevRTT = (1-β)*DevRTT +
             β*|SampleRTT-EstimatedRTT|

(typically, β = 0.25)
```

Then set timeout interval:

TimeoutInterval = EstimatedRTT + 4*DevRTT

TCP ACK generation [RFC 1122, RFC 2581]

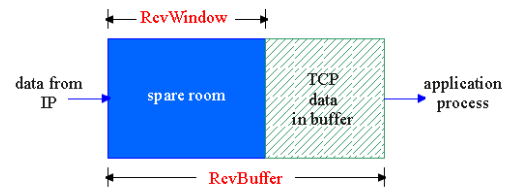
Event at Receiver	TCP Receiver action
Arrival of in-order segment with expected seq #. All data up to expected seq # already ACKed	Delayed ACK. Wait up to 500ms for next segment. If no next segment, send ACK
Arrival of in-order segment with expected seq #. One other segment has ACK pending	Immediately send single cumulative ACK, ACKing both in-order segments
Arrival of out-of-order segment higher-than-expect seq. # . Gap detected	Immediately send duplicate ACK, indicating seq. # of next expected byte
Arrival of segment that partially or completely fills gap	Immediate send ACK, provided that segment starts at lower end of gap

Fast Retransmit

- Time-out period often relatively long:
  - long delay before resending lost packet
- Detect lost segments via duplicate ACKs.
  - Sender often sends many segments back-to-back
  - If segment is lost, there will likely be many duplicate ACKs.
- If sender receives 3 ACKs for the same data, it supposes that segment after ACKed data was lost:
  - fast retransmit: resend segment before timer expires

TCP Flow Control

- receive side of TCP connection has a receive buffer:

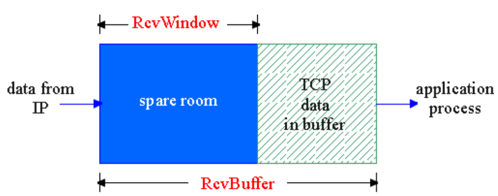


- app process may be slow at reading from buffer

flow control
sender won't overflow receiver's buffer by transmitting too much, too fast

- speed-matching service: matching the send rate to the receiving app's drain rate

TCP Flow control: how it works



(Suppose TCP receiver discards out-of-order segments)

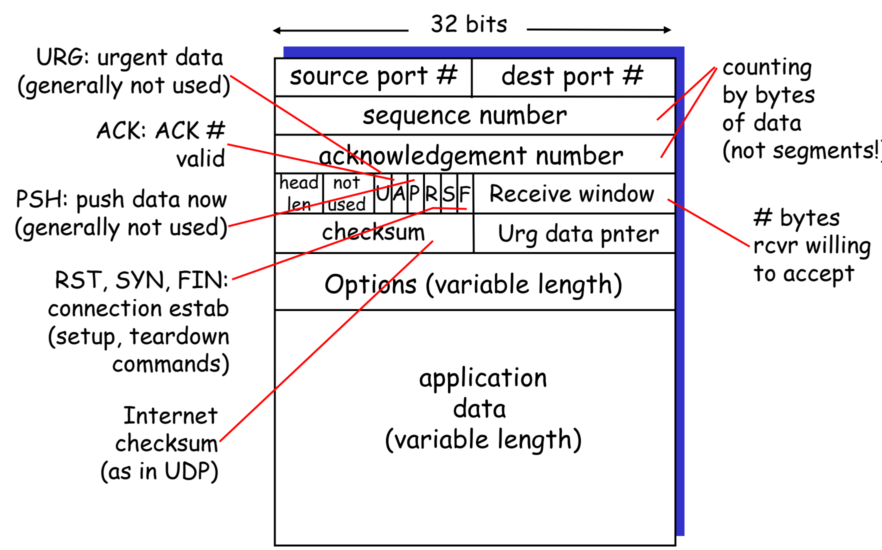
- spare room in buffer

= **RcvWindow**

= **RcvBuffer-[LastByteRcvd - LastByteRead]**

- Rcvr advertises spare room by including value of **RcvWindow** in segments
- Sender limits unACKed data to **RcvWindow**
  - guarantees receive buffer doesn't overflow

TCP segment structure



Recall that TCP **now** gives:

- ☑ reliable data transfer (make sure the message gets through)
- ☑ flow control (don't overwhelm the receiver)
- ❒ congestion control (don't overwhelm the network)

UDP gives:

- No reliability, no packet ordering
- No flow control
- …

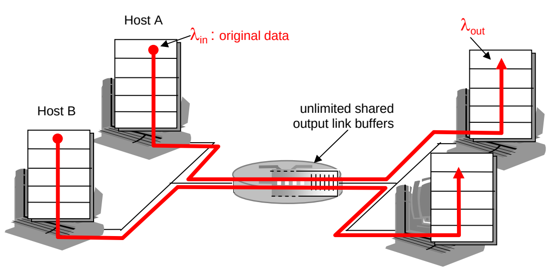
Principles of Congestion Control

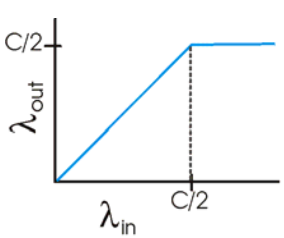
Congestion:

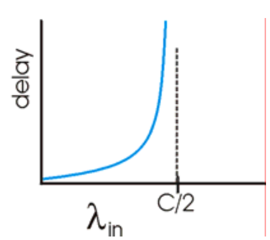
- informally: "too many sources sending too much data too fast for *network* to handle"
- different from flow control!
- is expressed as:
  - lost packets (buffer overflow at routers)
  - long delays (queueing in router buffers)
- a top-10 network problem!
- packet retransmission treats the symptom not the cause (even worsens the cause!)

Causes/costs of congestion: scenario 1

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



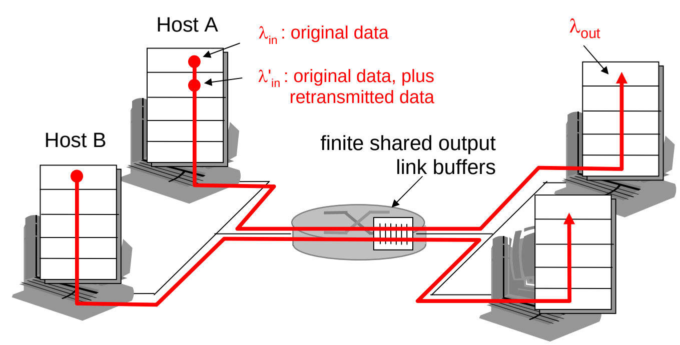




- large delays when congested
- maximum achievable throughput

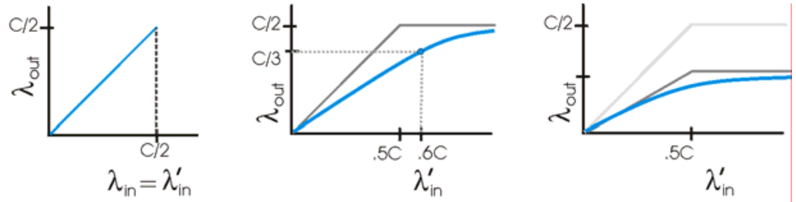
Causes/costs of congestion: scenario 2

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



Causes/costs of congestion: scenario 2

- always: $\lambda_{in} = \lambda_{out}$ (goodput), $\lambda'_{in} > \lambda_{out}$
- "perfect" retransmission (no router overhead) only when loss
- retransmission of delayed (not lost) packet makes λ'_{in} even larger (than perfect case) for same λ_{out}



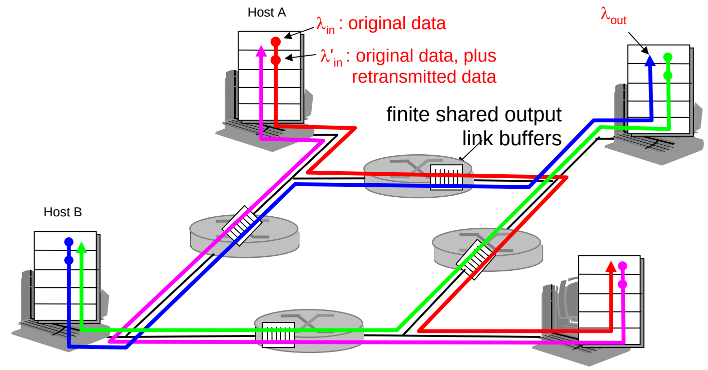
"costs" of congestion:

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

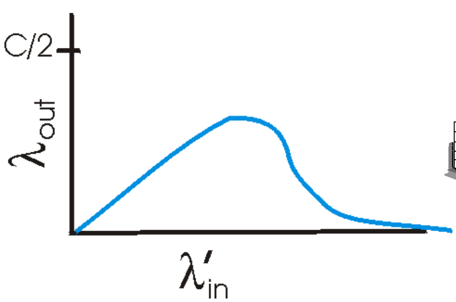
Causes/costs of congestion: scenario 3

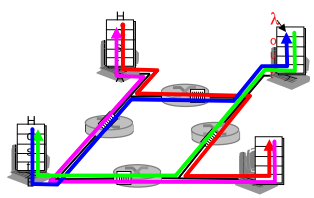
- four senders
- multihop paths
- timeout/retransmit

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



Causes/costs of congestion: scenario 3





Another "cost" of congestion:

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

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, ATM)
  - explicit rate that sender should send at

Exercise 2

Clock synchronization:

*There exists γ, $\boldsymbol{t_0}$, v, **a** and **b** such that $\forall \boldsymbol{t} \geq \boldsymbol{t_0}$:*

- ***Agreement***. For any correct nodes *p*, *q* :
  "Clocks *close* to **each other**"
- ***Validity***. For every correct node *p* :
  "Clocks *close* to **real time**"