

Chapter 3- parte B 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

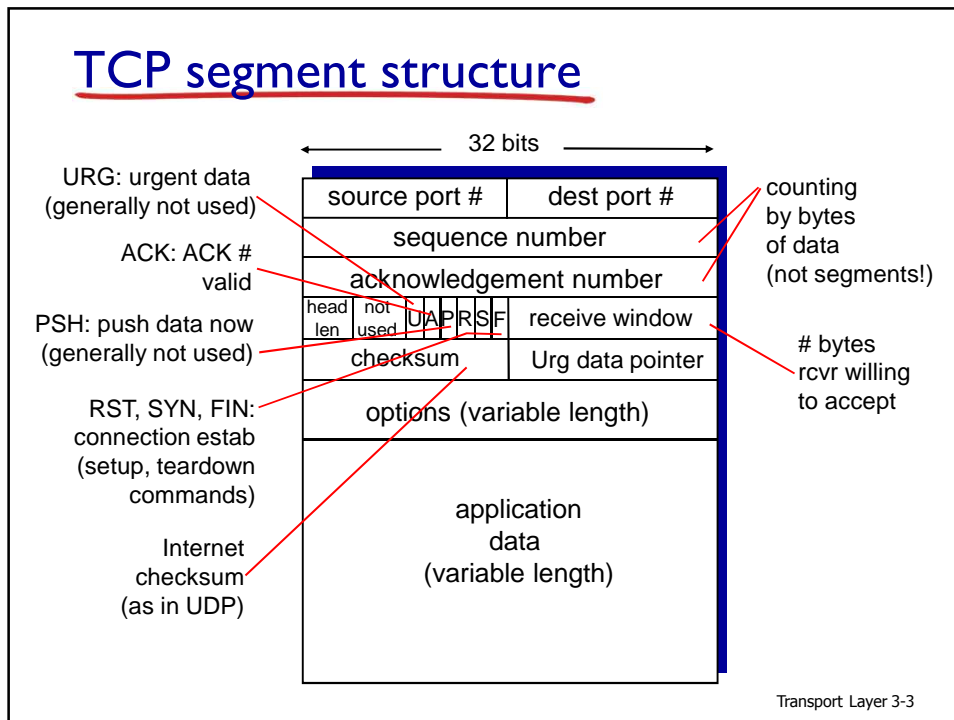
Transport Layer 3-1

TCP: Overview RFCs: 793, 1122, 1323, 2018, 2581

- ❖ **point-to-point:**
 - one sender, one receiver
- ❖ **reliable, in-order byte stream:**
 - no “message boundaries”
- ❖ **pipelined:**
 - TCP congestion and flow control set window size
- ❖ **full duplex data:**
 - bi-directional data flow in same connection
 - MSS: maximum segment size
- ❖ **connection-oriented:**
 - handshaking (exchange of control msgs) inits sender, receiver state before data exchange
- ❖ **flow controlled:**
 - sender will not overwhelm receiver

Transport Layer 3-2

TCP segment structure



TCP seq. numbers, ACKs

sequence numbers:

- byte stream "number" of first byte in segment's data

acknowledgements:

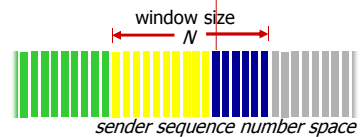
- seq # of next byte expected from other side
- cumulative ACK

Q: how receiver handles out-of-order segments

- A: TCP spec doesn't say, - up to implementor

outgoing segment from sender

source port #	dest port #
sequence number	
acknowledgement number	
	rwnd
checksum	urg pointer



sent ACKed

sent, not-yet ACKed ("in-flight")

usable but not yet sent

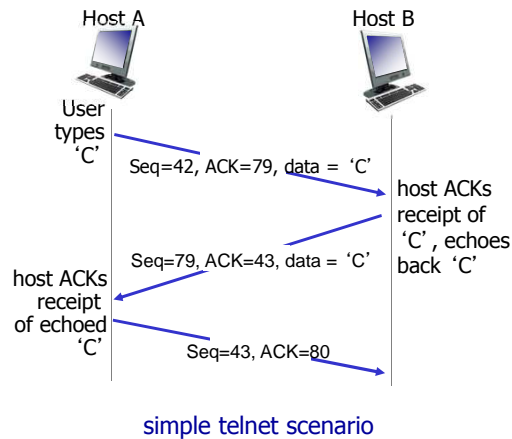
not usable

incoming segment to sender

source port #	dest port #
sequence number	
acknowledgement number	
A	rwnd
checksum	urg pointer

Transport Layer 3-4

TCP seq. numbers, ACKs



Transport Layer 3-5

TCP round trip time, timeout

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?

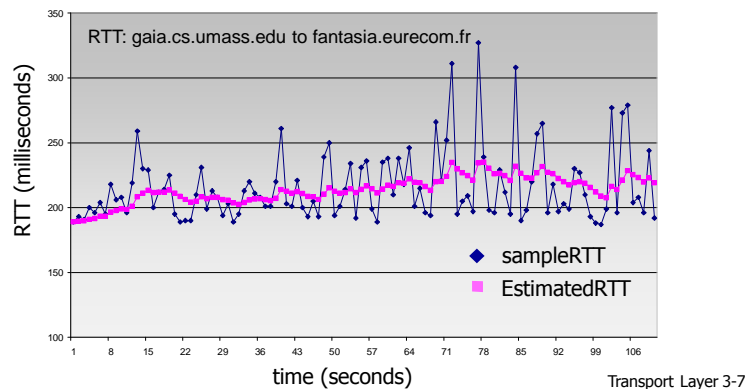
- ❖ **SampleRTT**: measured time from segment transmission until ACK receipt
 - ignore retransmissions
- ❖ **SampleRTT** will vary, want estimated RTT "smoother"
 - average several *recent* measurements, not just current **SampleRTT**

Transport Layer 3-6

TCP round trip time, timeout

$$\text{EstimatedRTT} = (1 - \alpha) * \text{EstimatedRTT} + \alpha * \text{SampleRTT}$$

- ❖ exponential weighted moving average
- ❖ influence of past sample decreases exponentially fast
- ❖ typical value: $\alpha = 0.125$



TCP round trip time, timeout

- ❖ **timeout interval:** **EstimatedRTT** plus “safety margin”
 - large variation in **EstimatedRTT** -> larger safety margin
- ❖ estimate SampleRTT deviation from EstimatedRTT:

$$\text{DevRTT} = (1 - \beta) * \text{DevRTT} + \beta * |\text{SampleRTT} - \text{EstimatedRTT}|$$

(typically, $\beta = 0.25$)

$$\text{TimeoutInterval} = \text{EstimatedRTT} + 4 * \text{DevRTT}$$



↑
↑
 estimated RTT “safety margin”

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

Transport Layer 3-9

TCP reliable data transfer

- ❖ TCP creates rdt service on top of IP's unreliable service
 - pipelined segments
 - cumulative acks
 - single retransmission timer
 - ❖ retransmissions triggered by:
 - timeout events
 - duplicate acks
- let's initially consider simplified TCP sender:
- ignore duplicate acks
 - ignore flow control, congestion control

Transport Layer 3-10

TCP sender events:

data rcvd from app:

- ❖ create segment with seq #
- ❖ seq # is byte-stream number of first data byte in segment
- ❖ start timer if not already running
 - think of timer as for oldest unacked segment
 - expiration interval: `TimeoutInterval`

timeout:

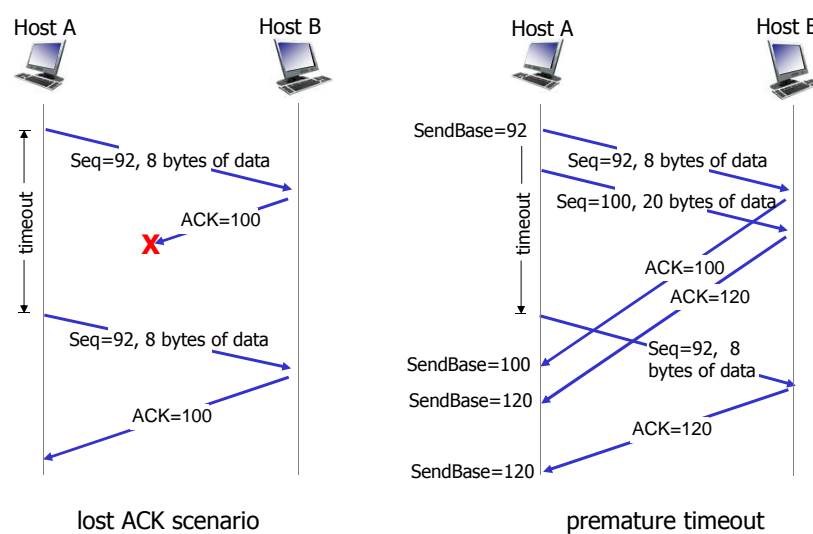
- ❖ retransmit segment that caused timeout
- ❖ restart timer

ack rcvd:

- ❖ if ack acknowledges previously unacked segments
 - update what is known to be ACKed
 - start timer if there are still unacked segments

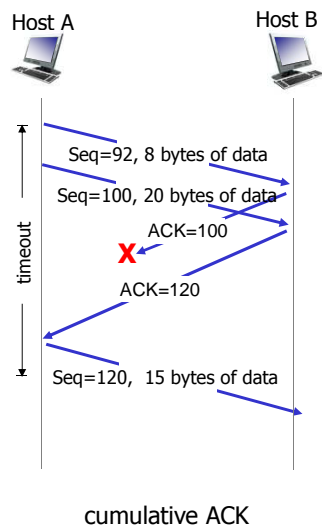
Transport Layer 3-11

TCP: retransmission scenarios



Transport Layer 3-12

TCP: retransmission scenarios



Transport Layer 3-13

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

Transport Layer 3-14

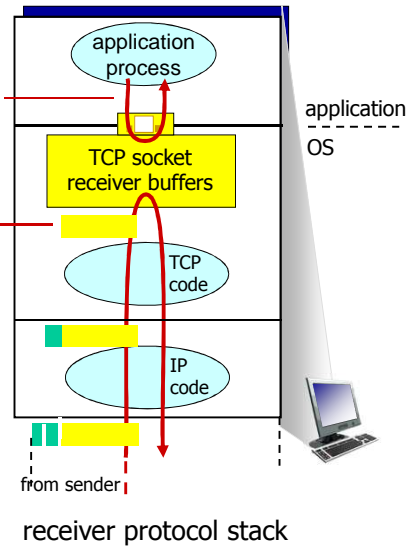
TCP flow control

application may remove data from TCP socket buffers

... slower than TCP receiver is delivering (sender is sending)

flow control

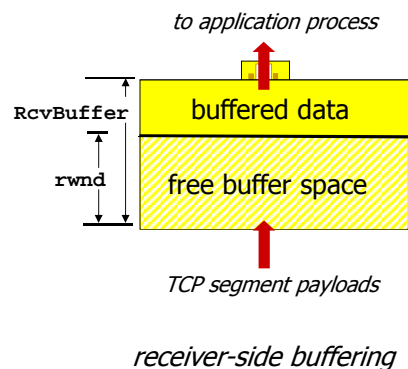
receiver controls sender, so sender won't overflow receiver's buffer by transmitting too much, too fast



Transport Layer 3-15

TCP flow control

- ❖ receiver “advertises” free buffer space by including **rwnd** value in TCP header of receiver-to-sender segments
 - **RcvBuffer** size set via socket options (typical default is 4096 bytes)
 - many operating systems autoadjust **RcvBuffer**
- ❖ sender limits amount of unacked (“in-flight”) data to receiver’s **rwnd** value
- ❖ guarantees receive buffer will not overflow



Transport Layer 3-16

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

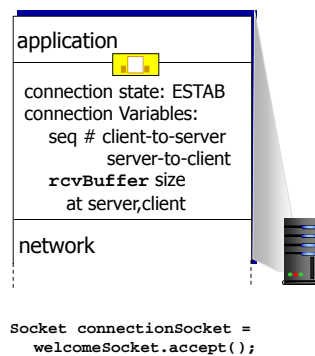
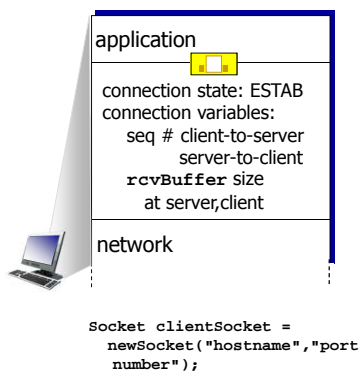
3.7 TCP congestion control

Transport Layer 3-17

Connection Management

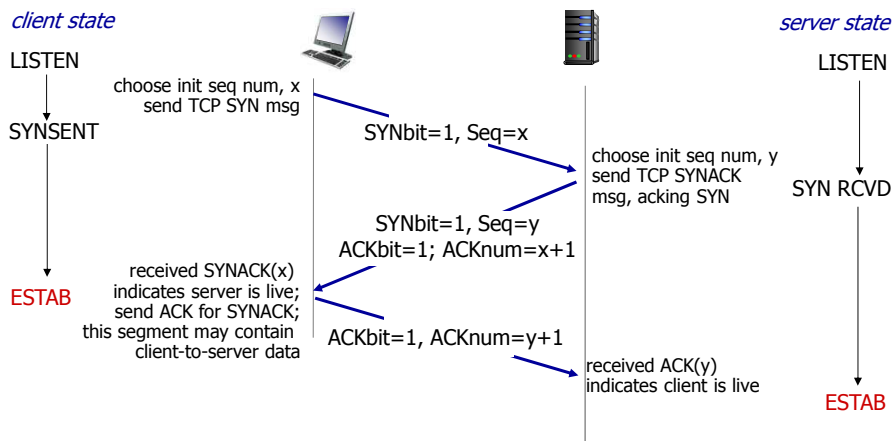
before exchanging data, sender/receiver “handshake”:

- ❖ agree to establish connection (each knowing the other willing to establish connection)
- ❖ agree on connection parameters



Transport Layer 3-18

TCP 3-way handshake

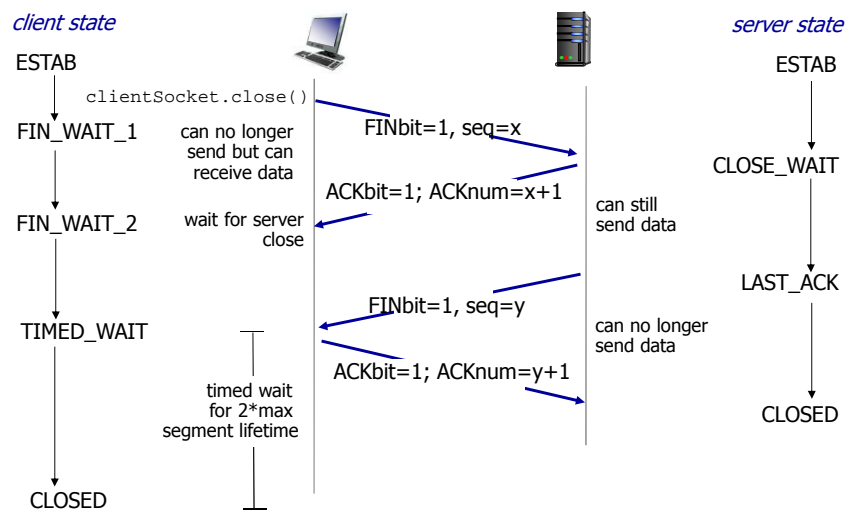


TCP: closing a connection

- ❖ client, server each close their side of connection
 - send TCP segment with FIN bit = 1
- ❖ respond to received FIN with ACK
 - on receiving FIN, ACK can be combined with own FIN
- ❖ simultaneous FIN exchanges can be handled

Transport Layer 3-20

TCP: closing a connection



Transport Layer 3-21

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

Transport Layer 3-22

Principles of congestion control

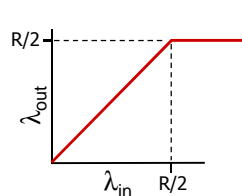
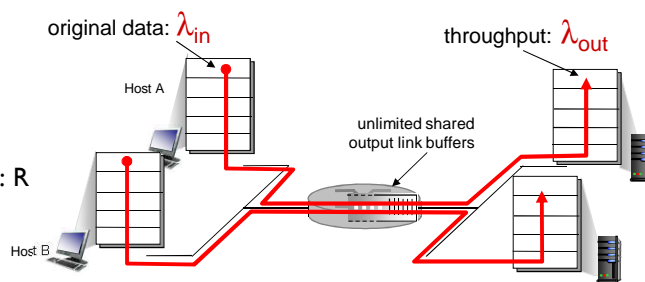
congestion:

- ❖ 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!

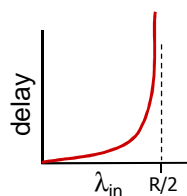
Transport Layer 3-23

Causes/costs of congestion: scenario I

- ❖ two senders, two receivers
- ❖ one router, infinite buffers
- ❖ output link capacity: R
- ❖ no retransmission



- ❖ maximum per-connection throughput: $R/2$

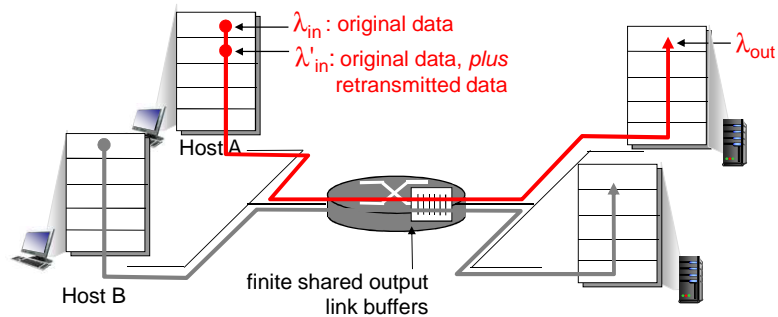


- ❖ large delays as arrival rate, λ_{in} , approaches capacity

Transport Layer 3-24

Causes/costs of congestion: scenario 2

- ❖ one router, *finite* buffers
- ❖ sender retransmission of timed-out packet
 - application-layer input = application-layer output: $\lambda_{in} = \lambda_{out}$
 - transport-layer input includes *retransmissions* : $\lambda'_{in} \geq \lambda_{in}$

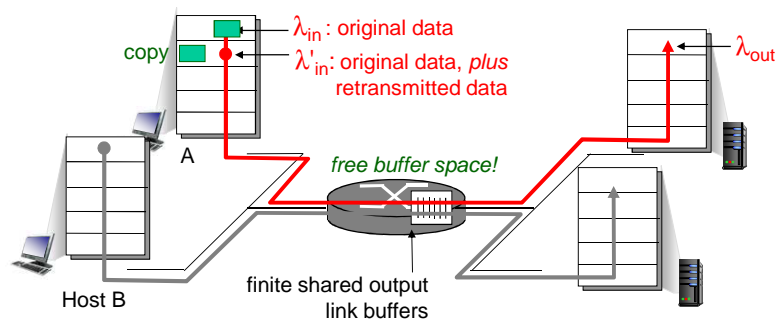
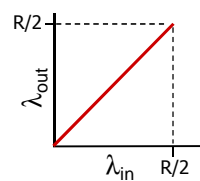


Transport Layer 3-25

Causes/costs of congestion: scenario 2

idealization: perfect knowledge

- ❖ sender sends only when router buffers available



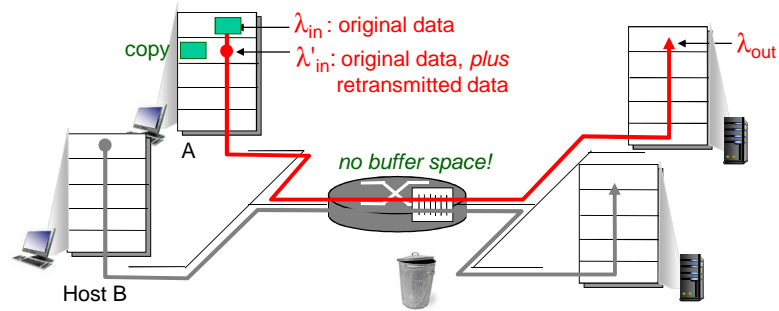
Transport Layer 3-26

Causes/costs of congestion: scenario 2

Idealization: *known loss*

packets can be lost, dropped at router due to full buffers

- ❖ sender only resends if packet *known* to be lost

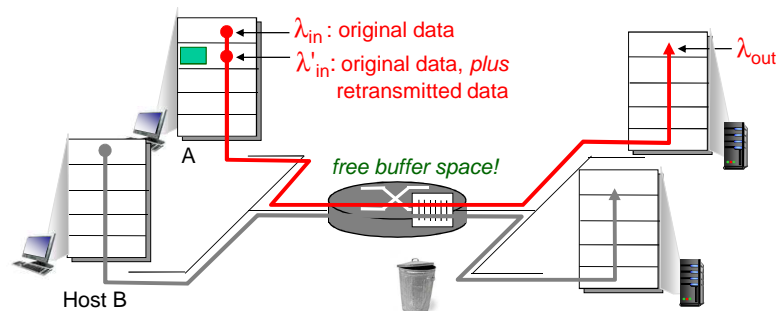
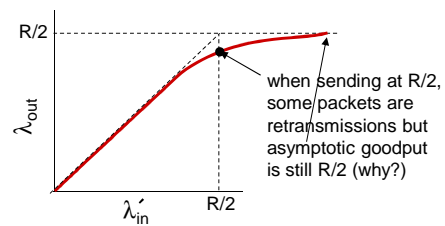


Causes/costs of congestion: scenario 2

Idealization: *known loss*

packets can be lost, dropped at router due to full buffers

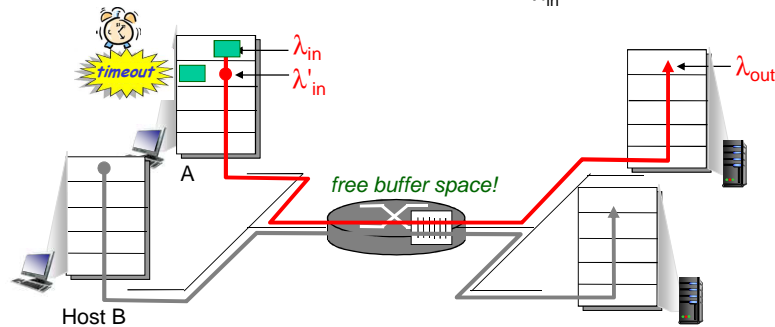
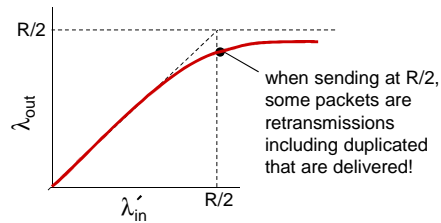
- ❖ sender only resends if packet *known* to be lost



Causes/costs of congestion: scenario 2

Realistic: *duplicates*

- ❖ packets can be lost, dropped at router due to full buffers
- ❖ sender times out prematurely, sending *two* copies, both of which are delivered

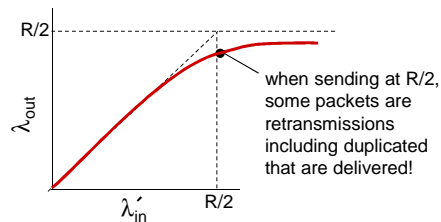


Transport Layer 3-29

Causes/costs of congestion: scenario 2

Realistic: *duplicates*

- ❖ packets can be lost, dropped at router due to full buffers
- ❖ sender times out prematurely, sending *two* copies, both of which are delivered

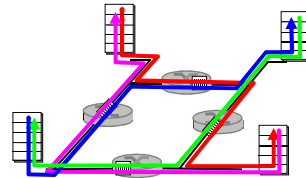
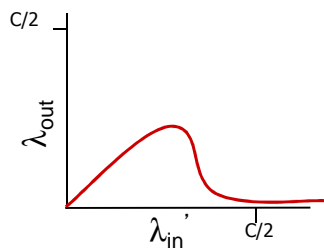


"costs" of congestion:

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

Transport Layer 3-30

Causes/costs of congestion: scenario 3



another “cost” of congestion:

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

Transport Layer 3-31

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 for sender to send at

Transport Layer 3-32

Chapter 3 outline

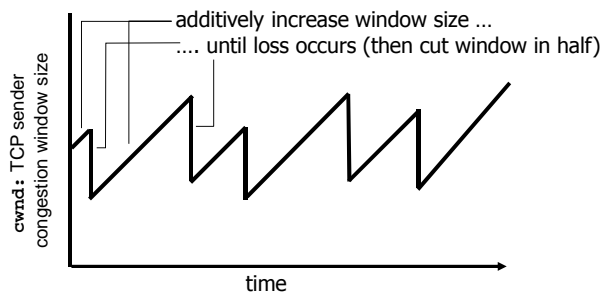
- 3.1 transport-layer services
- 3.2 multiplexing and demultiplexing
- 3.3 connectionless transport: UDP
- 3.4 principles of reliable data transfer
- 3.5 connection-oriented transport: TCP
 - segment structure
 - reliable data transfer
 - flow control
 - connection management
- 3.6 principles of congestion control
- 3.7 TCP congestion control

Transport Layer 3-33

TCP congestion control: additive increase multiplicative decrease

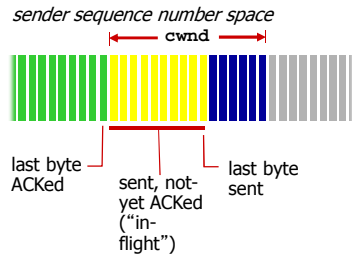
- ❖ *approach*: sender increases transmission rate (window size), probing for usable bandwidth, until loss occurs
 - *additive increase*: increase `cwnd` by 1 MSS every RTT until loss detected
 - *multiplicative decrease*: cut `cwnd` in half after loss

AIMD saw tooth behavior: probing for bandwidth



Transport Layer 3-34

TCP Congestion Control: details



- ❖ sender limits transmission:

$$\text{LastByteSent} - \text{LastByteAcked} \leq \text{cwnd}$$

- ❖ **cwnd** is dynamic, function of perceived network congestion

TCP sending rate:

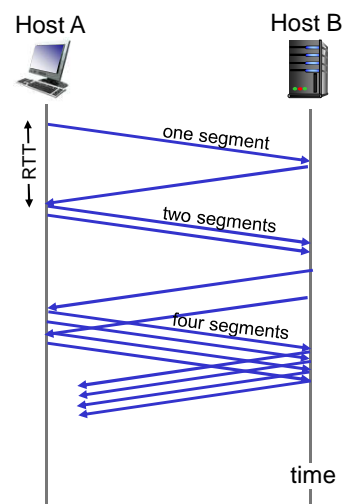
- ❖ *roughly*: send cwnd bytes, wait RTT for ACKS, then send more bytes

$$\text{rate} \approx \frac{\text{cwnd}}{\text{RTT}} \text{ bytes/sec}$$

Transport Layer 3-35

TCP Slow Start

- ❖ when connection begins, increase rate exponentially until first loss event:
 - initially **cwnd** = 1 MSS
 - double **cwnd** every RTT
 - done by incrementing **cwnd** for every ACK received
- ❖ **summary**: initial rate is slow but ramps up exponentially fast



Transport Layer 3-36

TCP: detecting, reacting to loss

- ❖ loss indicated by timeout:
 - `cwnd` set to 1 MSS;
 - window then grows exponentially (as in slow start) to threshold, then grows linearly
- ❖ loss indicated by 3 duplicate ACKs: TCP RENO
 - dup ACKs indicate network capable of delivering some segments
 - `cwnd` is cut in half window then grows linearly
- ❖ TCP Tahoe always sets `cwnd` to 1 (timeout or 3 duplicate acks)

Transport Layer 3-37

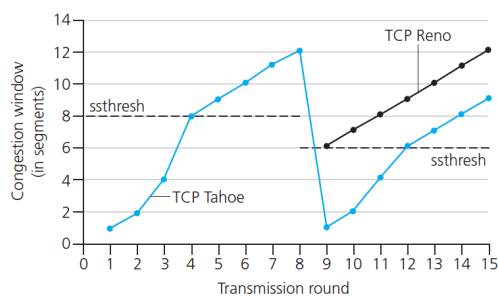
TCP: switching from slow start to CA

Q: when should the exponential increase switch to linear?

A: when `cwnd` gets to 1/2 of its value before timeout.

Implementation:

- ❖ variable `ssthresh`
- ❖ on loss event, `ssthresh` is set to 1/2 of `cwnd` just before loss event

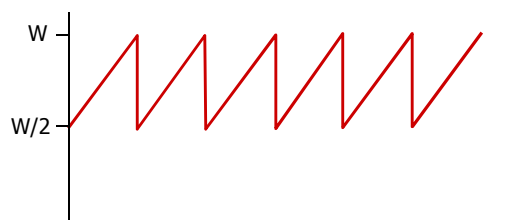


Transport Layer 3-38

TCP throughput

- ❖ avg. TCP thrupt as function of window size, RTT?
 - ignore slow start, assume always data to send
- ❖ **W: window size** (measured in bytes) where loss occurs
 - avg. window size (# in-flight bytes) is $\frac{3}{4} W$
 - avg. thrupt is $3/4W$ per RTT

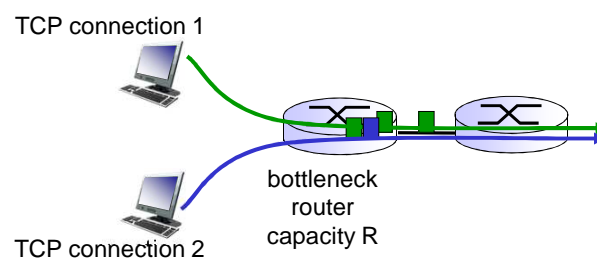
$$\text{avg TCP thrupt} = \frac{3}{4} \frac{W}{\text{RTT}} \text{ bytes/sec}$$



Transport Layer 3-39

TCP Fairness

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



Transport Layer 3-40