

Introduction to Computer Networks

Transport Layer Overview (§6.1.2-6.1.4)



Computer Science & Engineering

UNIVERSITY of WASHINGTON

Transport Layer Services

- Provide different kinds of data delivery across the network to applications

	Unreliable	Reliable
Messages	Datagrams (UDP)	
Bytestream		Streams (TCP)

Comparison of Internet Transports

- TCP is full-featured, UDP is a glorified packet

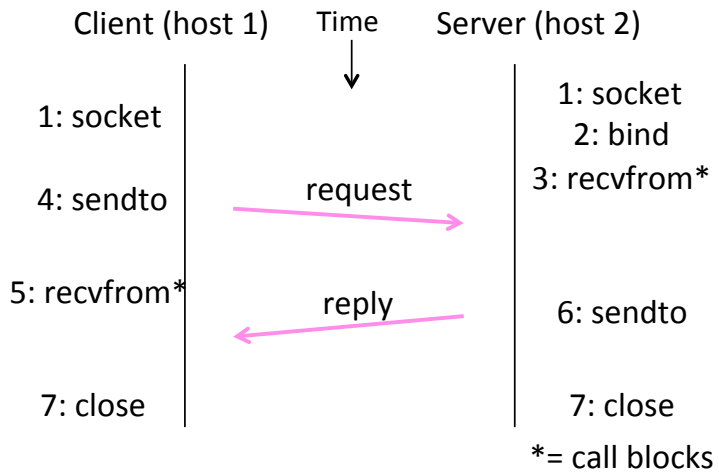
TCP (Streams)	UDP (Datagrams)
Connections	Datagrams
Bytes are delivered once, reliably, and in order	Messages may be lost, reordered, duplicated
Arbitrary length content	Limited message size
Flow control matches sender to receiver	Can send regardless of receiver state
Congestion control matches sender to network	Can send regardless of network state

User Datagram Protocol (UDP)

- Used by apps that don't want reliability or bytestreams
 - Voice-over-IP (unreliable)
 - DNS, RPC (message-oriented)
 - DHCP (bootstrapping)

(If application wants reliability and messages then it has work to do!)

Datagram Sockets

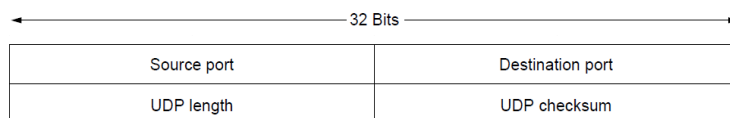


CSE 461 University of Washington

17

UDP Header

- Uses ports to identify sending and receiving application processes
- Datagram length up to 64K
- Checksum (16 bits) for reliability



CSE 461 University of Washington

19

Introduction to Computer Networks

Connection Establishment (§6.5.6, §6.5.7, §6.2.3)



Computer Science & Engineering

UNIVERSITY *of* WASHINGTON

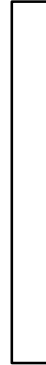
Connection Establishment

- Both sender and receiver must be ready before we start the transfer of data
 - Need to agree on a set of parameters
 - e.g., the Maximum Segment Size (MSS)
- This is signaling
 - It sets up state at the endpoints
 - Like “dialing” for a telephone call

Three-Way Handshake

- Used in TCP; opens connection for data in both directions
- Each side probes the other with a fresh Initial Sequence Number (ISN)
 - Sends on a SYNchronize segment
 - Echo on an ACKnowledge segment
- Chosen to be robust even against delayed duplicates

Active party
(client)



Passive party
(server)



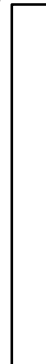
CSE 461 University of Washington

24

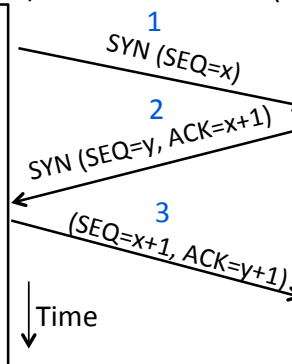
Three-Way Handshake (2)

- Three steps:
 - Client sends SYN(x)
 - Server replies with SYN(y)ACK(x+1)
 - Client replies with ACK(y+1)
 - SYNs are retransmitted if lost
- Sequence and ack numbers carried on further segments

Active party
(client)



Passive party
(server)

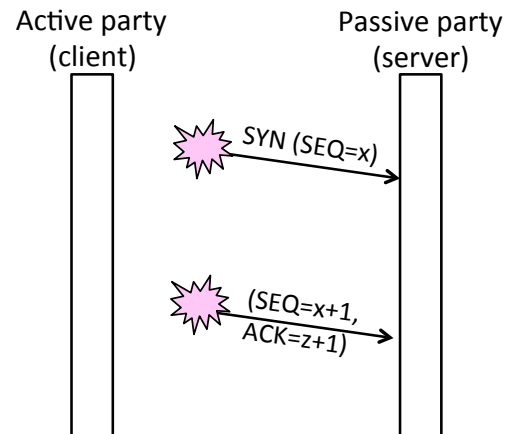


CSE 461 University of Washington

25

Three-Way Handshake (3)

- Suppose delayed, duplicate copies of the SYN and ACK arrive at the server!
 - Improbable, but anyhow ...



Connection Release

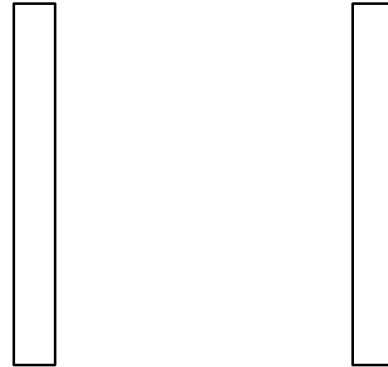
- Orderly release by both parties when done
 - Delivers all pending data and “hangs up”
 - Cleans up state in sender and receiver
- Key problem is to provide reliability while releasing
 - TCP uses a “symmetric” close in which both sides shutdown independently

TCP Connection Release

- Two steps:
 - Active sends FIN(x), passive ACKs
 - Passive sends FIN(y), active ACKs
 - FINs are retransmitted if lost
- Each FIN/ACK closes one direction of data transfer

Active party

Passive party



CSE 461 University of Washington

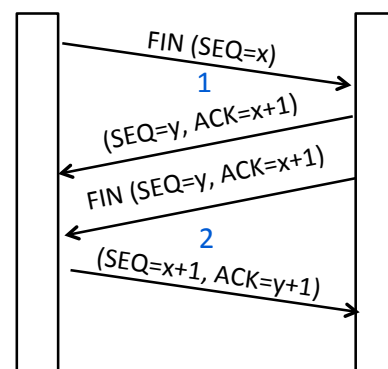
36

TCP Connection Release (2)

- Two steps:
 - Active sends FIN(x), passive ACKs
 - Passive sends FIN(y), active ACKs
 - FINs are retransmitted if lost
- Each FIN/ACK closes one direction of data transfer

Active party

Passive party

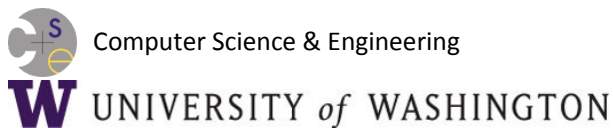


CSE 461 University of Washington

37

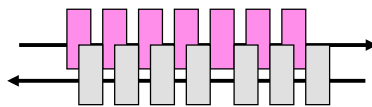
Introduction to Computer Networks

Sliding Windows (§3.4, §6.5.8)



Sliding Window

- Generalization of stop-and-wait
 - Allows W packets to be outstanding
 - Can send W packets per RTT ($=2D$)



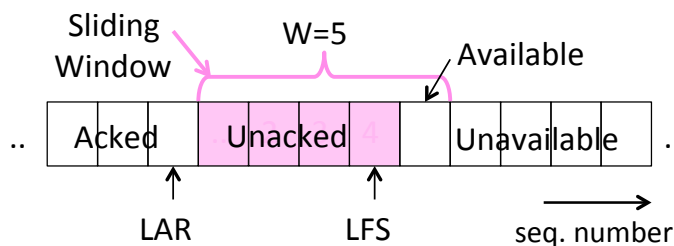
- Pipelining improves performance
- Need $W=2BD$ to fill network path

Sliding Window Protocol

- Many variations, depending on how buffers, acknowledgements, and retransmissions are handled
- Go-Back-N »
 - Simplest version, can be inefficient
- Selective Repeat »
 - More complex, better performance

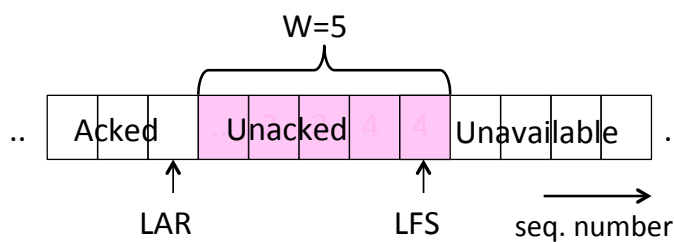
Sliding Window – Sender

- Sender buffers up to W segments until they are acknowledged
 - $LFS = \text{LAST FRAME SENT}$, $LAR = \text{LAST ACK REC'D}$
 - Sends while $LFS - LAR \leq W$



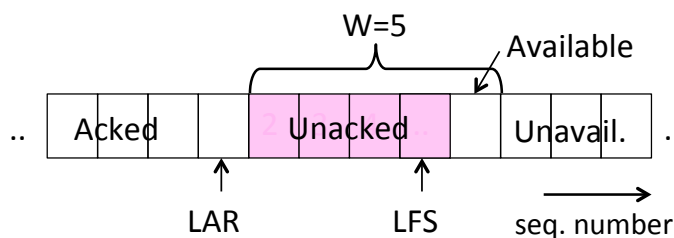
Sliding Window – Sender (2)

- Transport accepts another segment of data from the Application ...
 - Transport sends it (as $LFS-LAR \rightarrow 5$)



Sliding Window – Sender (3)

- Next higher ACK arrives from peer...
 - Window advances, buffer is freed
 - $LFS-LAR \rightarrow 4$ (can send one more)



Sliding Window – Go-Back-N

- Receiver keeps only a single packet buffer for the next segment
 - State variable, LAS = LAST ACK SENT
- On receive:
 - If seq. number is LAS+1, accept and pass it to app, update LAS, send ACK
 - Otherwise discard (as out of order)

Sliding Window – Selective Repeat

- Receiver passes data to app in order, and buffers out-of-order segments to reduce retransmissions
- ACK conveys highest in-order segment, plus hints about out-of-order segments
- TCP uses a selective repeat design; we'll see the details later

Sliding Window – Selective Repeat (2)

- Buffers W segments, keeps state variable, $LAS = \text{LAST ACK SENT}$
- On receive:
 - Buffer segments $[LAS+1, LAS+W]$
 - Pass up to app in-order segments from $LAS+1$, and update LAS
 - Send ACK for LAS regardless

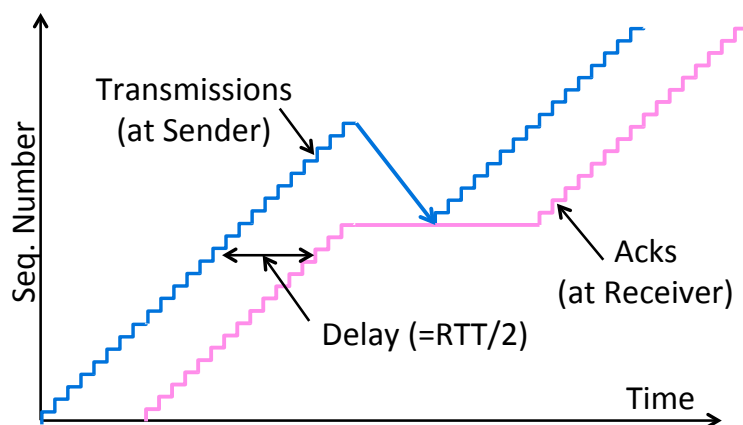
Sliding Window – Retransmissions

- Go-Back-N sender uses a single timer to detect losses
 - On timeout, resends buffered packets starting at $LAR+1$
- Selective Repeat sender uses a timer per unacked segment to detect losses
 - On timeout for segment, resend it
 - Hope to resend fewer segments

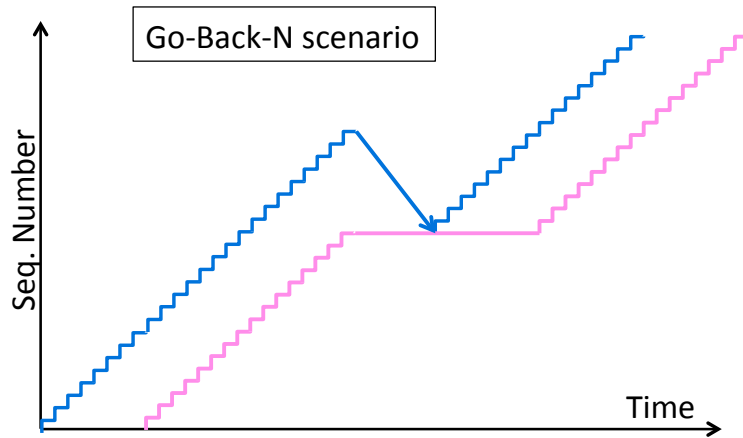
Sequence Numbers

- Need more than 0/1 for Stop-and-Wait ...
 - But how many?
- For Selective Repeat, need W numbers for packets, plus W for acks of earlier packets
 - $2W$ seq. numbers
 - Fewer for Go-Back- N ($W+1$)
- Typically implement seq. number with an N -bit counter that wraps around at $2^N - 1$
 - E.g., $N=8$: ..., 253, 254, 255, 0, 1, 2, 3, ...

Sequence Time Plot



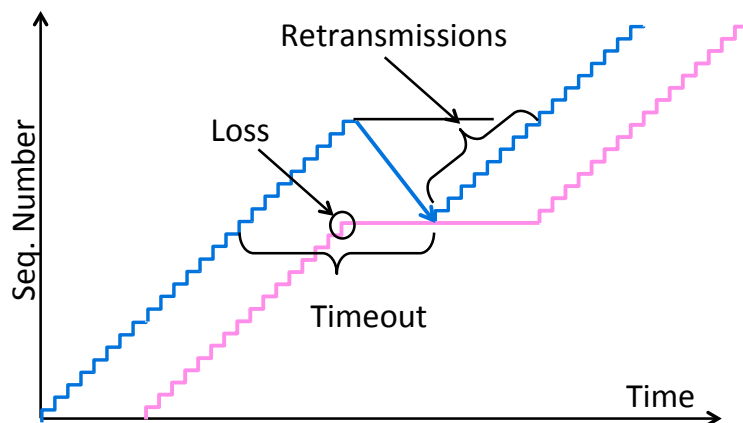
Sequence Time Plot (2)



CSE 461 University of Washington

61

Sequence Time Plot (3)

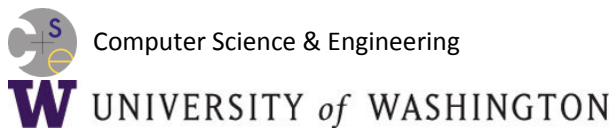


CSE 461 University of Washington

62

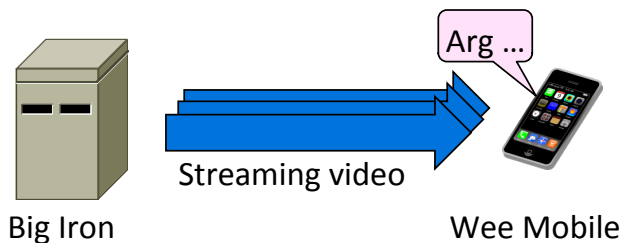
Introduction to Computer Networks

Flow Control (§6.5.8)



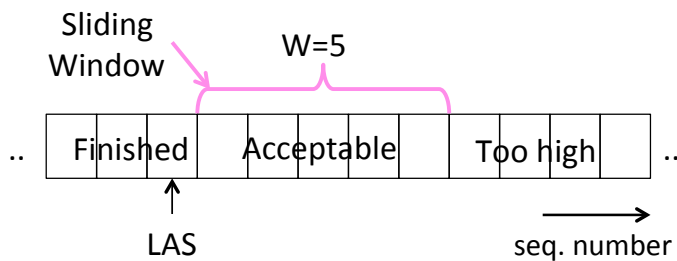
Problem

- Sliding window uses pipelining to keep the network busy
 - What if the receiver is overloaded?



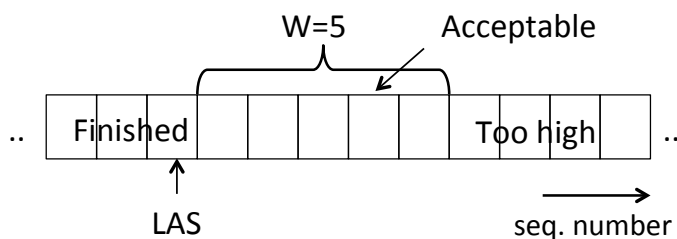
Sliding Window – Receiver

- Consider receiver with W buffers
 - LAS=LAST ACK SENT, app pulls in-order data from buffer with `recv()` call



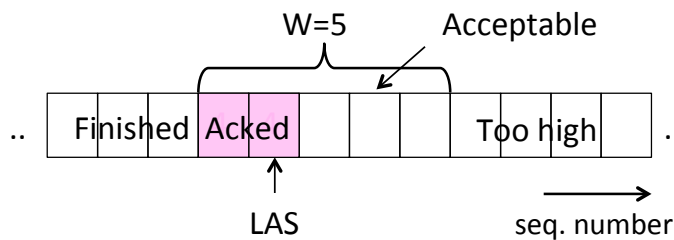
Sliding Window – Receiver (2)

- Suppose the next two segments arrive but app does not call `recv()`



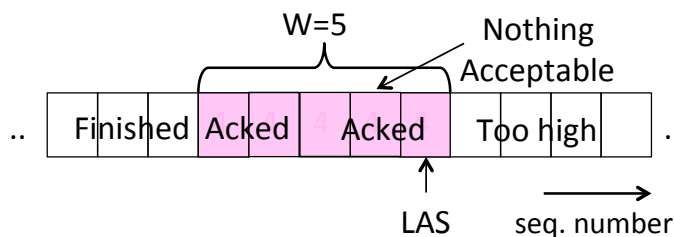
Sliding Window – Receiver (3)

- Suppose the next two segments arrive but app does not call `recv()`
 - LAS rises, but we can't slide window!



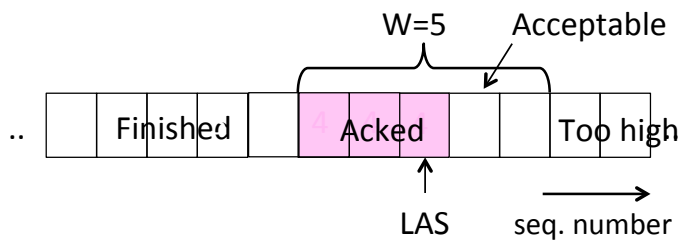
Sliding Window – Receiver (4)

- If further segments arrive (even in order) we can fill the buffer
 - Must drop segments until app `recvs!`



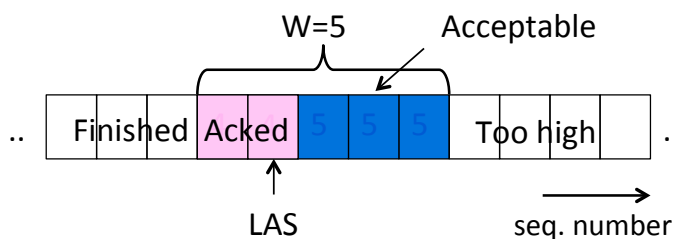
Sliding Window – Receiver (5)

- App recv() takes two segments
 - Window slides (pew)



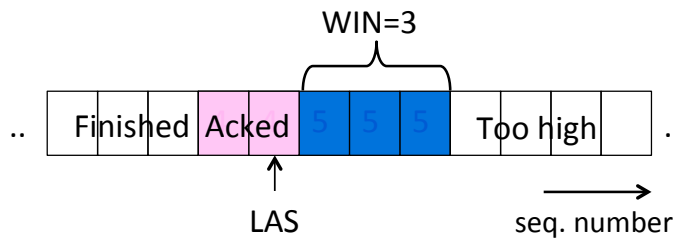
Flow Control

- Avoid loss at receiver by telling sender the available buffer space
 - WIN=#Acceptable, not W (from LAS)



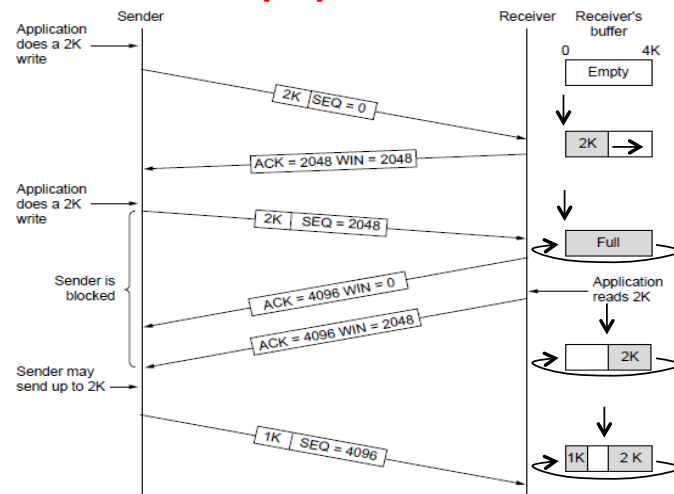
Flow Control (2)

- Sender uses the lower of the sliding window and flow control window (WIN) as the effective window size



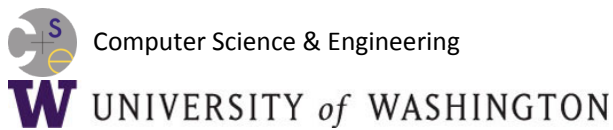
Flow Control (3)

- TCP-style example
 - SEQ/ACK sliding window
 - Flow control with WIN
 - $SEQ + \text{length} < ACK + WIN$
 - 4KB buffer at receiver
 - Circular buffer of bytes



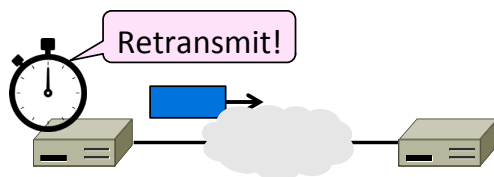
Introduction to Computer Networks

Retransmission Timeouts (§6.5.9)



Retransmissions

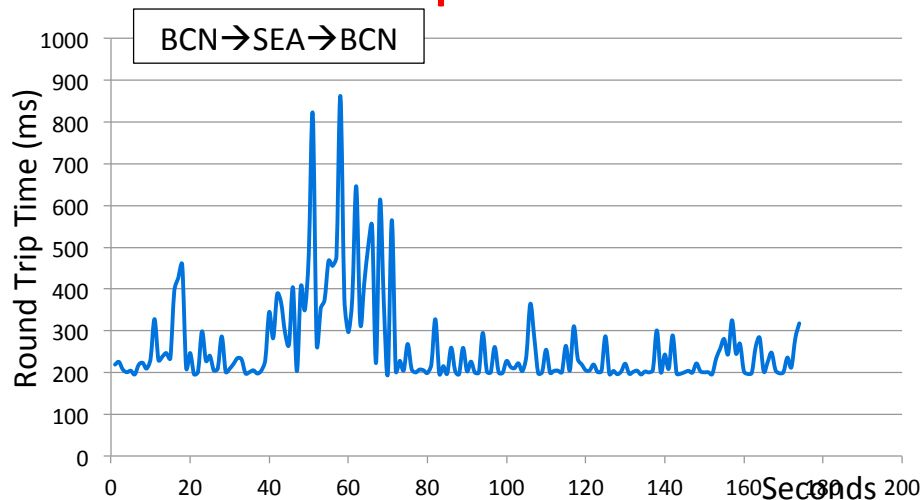
- With sliding window, the strategy for detecting loss is the timeout
 - Set timer when a segment is sent
 - Cancel timer when ack is received
 - If timer fires, retransmit data as lost



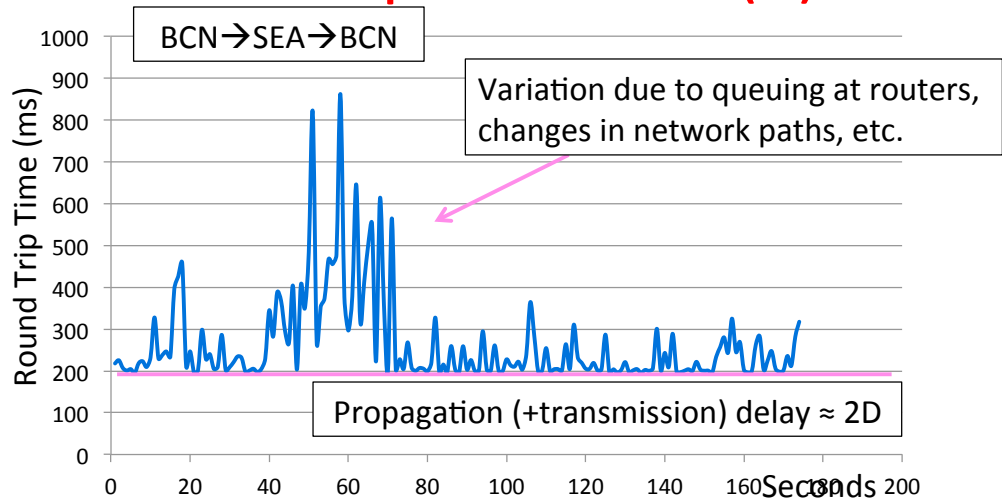
Timeout Problem

- Timeout should be “just right”
 - Too long wastes network capacity
 - Too short leads to spurious resends
 - But what is “just right”?
- Easy to set on a LAN (Link)
 - Short, fixed, predictable RTT
- Hard on the Internet (Transport)
 - Wide range, variable RTT

Example of RTTs



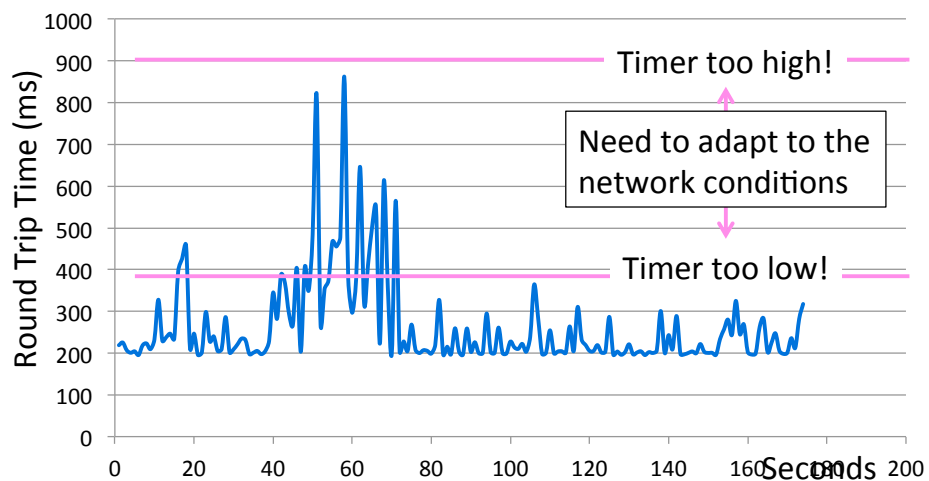
Example of RTTs (2)



CSE 461 University of Washington

79

Example of RTTs (3)



CSE 461 University of Washington

80

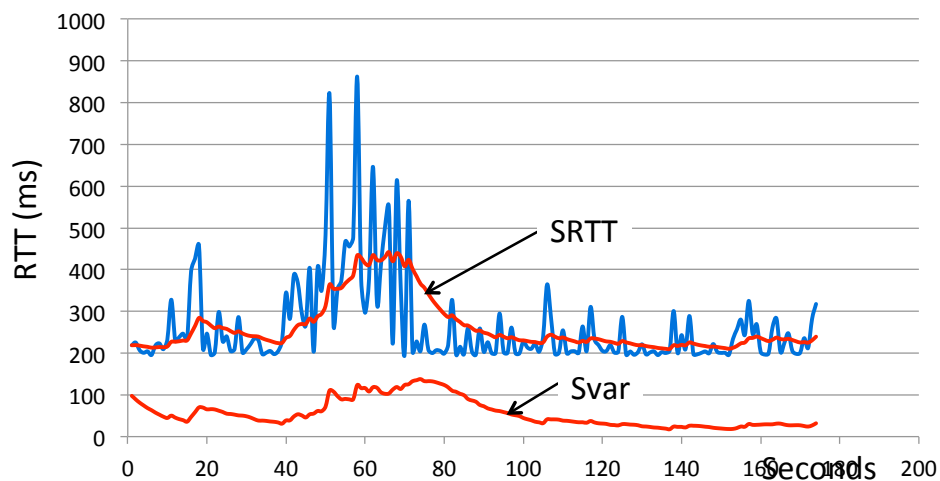
Adaptive Timeout

- Keep smoothed estimates of the RTT (1) and variance in RTT (2)
 - Update estimates with a moving average
 1. $SRTT_{N+1} = 0.9 * SRTT_N + 0.1 * RTT_{N+1}$
 2. $Svar_{N+1} = 0.9 * Svar_N + 0.1 * |RTT_{N+1} - SRTT_{N+1}|$
- Set timeout to a multiple of estimates
 - To estimate the upper RTT in practice
 - $TCP\ Timeout_N = SRTT_N + 4 * Svar_N$

CSE 461 University of Washington

81

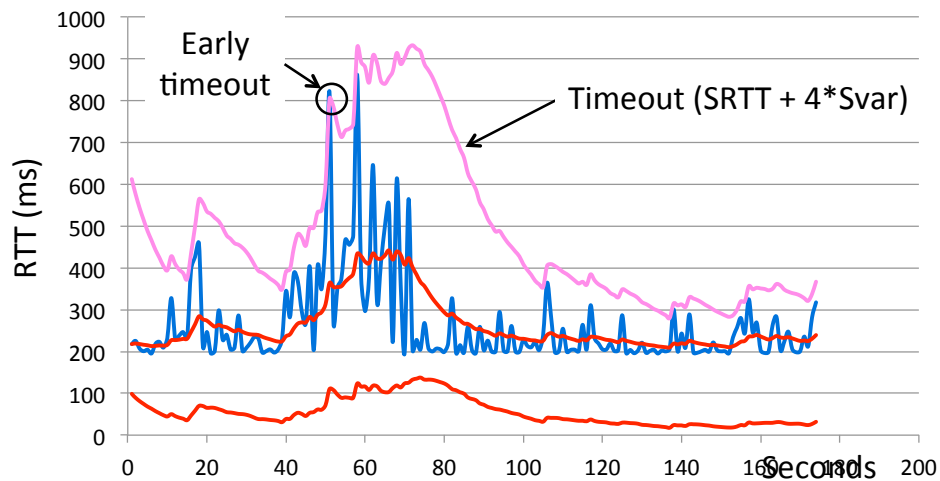
Example of Adaptive Timeout



CSE 461 University of Washington

82

Example of Adaptive Timeout (2)



CSE 461 University of Washington

83

Adaptive Timeout (2)

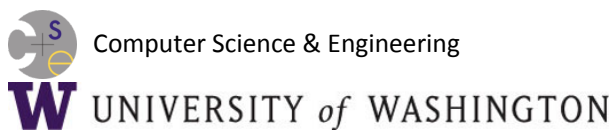
- Simple to compute, does a good job of tracking actual RTT
 - Little “headroom” to lower
 - Yet very few early timeouts
- Turns out to be important for good performance and robustness

CSE 461 University of Washington

84

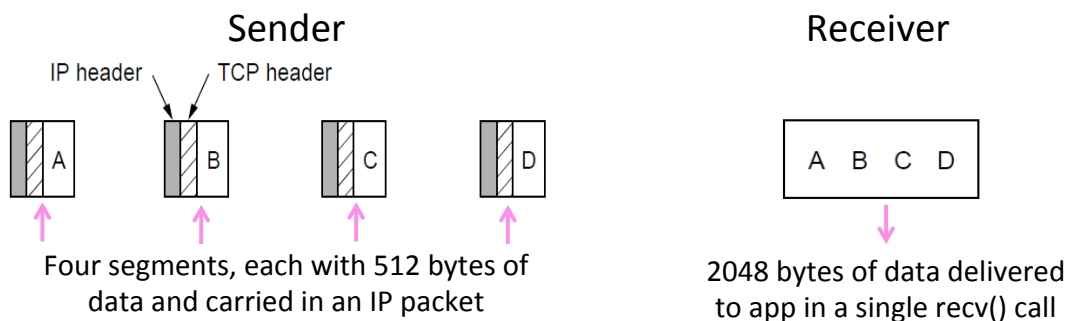
Introduction to Computer Networks

Transmission Control Protocol (TCP) (§6.5)



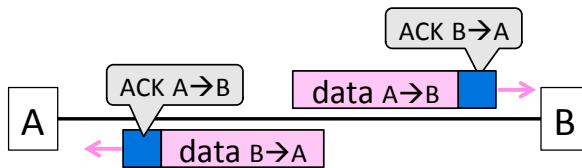
Reliable Bytestream

- Message boundaries not preserved from `send()` to `recv()`
 - But reliable and ordered (receive bytes in same order as sent)



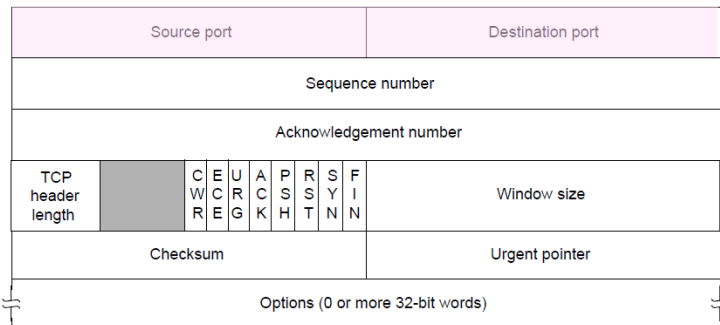
Reliable Bytestream (2)

- Bidirectional data transfer
 - Control information (e.g., ACK) piggybacks on data segments in reverse direction



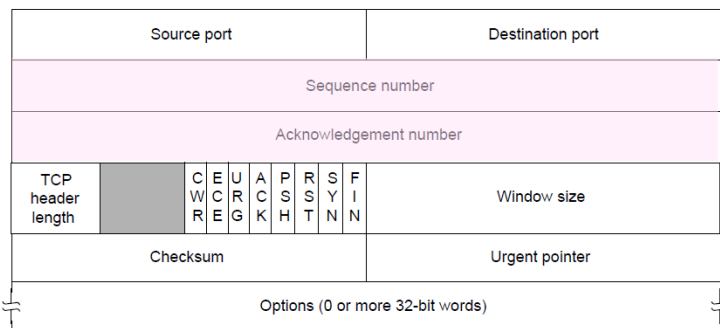
TCP Header (1)

- Ports identify apps (socket API)
 - 16-bit identifiers



TCP Header (2)

- SEQ/ACK used for sliding window
 - Selective Repeat, with byte positions

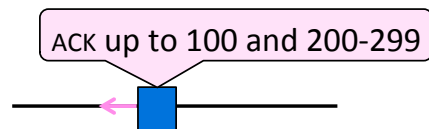


CSE 461 University of Washington

91

TCP Sliding Window – Receiver

- Cumulative ACK tells next expected byte sequence number (“LAS+1”)
- Optionally, selective ACKs (SACK) give hints for receiver buffer state
 - List up to 3 ranges of received bytes

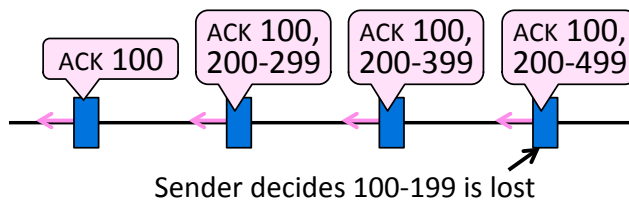


CSE 461 University of Washington

92

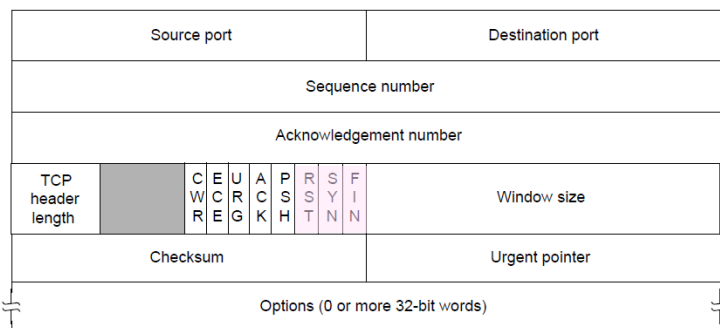
TCP Sliding Window – Sender

- Uses an adaptive retransmission timeout to resend data from $LAS+1$
- Uses heuristics to infer loss quickly and resend to avoid timeouts
 - “Three duplicate ACKs” treated as loss



TCP Header (3)

- SYN/FIN/RST flags for connections
 - Flag indicates segment is a SYN etc.



TCP Header (4)

- Window size for flow control
 - Relative to ACK, and in bytes

