CSC 458/2209 – Computer Networks

Handout # 9:
Transport Protocols

Professor Yashar Ganjali

Department of Computer Science

University of Toronto

yganjali@cs.toronto.edu

http://www.cs.toronto.edu/~yganjali

Announcements

- Programming assignment 1
  - Due: Friday, October 14th (5PM)
  - Please take academic integrity seriously

- This week
  - Chapter 5 of the textbook
  - Tutorial: programming assignment 1

- Next week: no tutorial
- Sample midterm and solutions on class web site

Midterm Exam

- Next week
  - Section L0101: Thu. Oct. 17th, 1-3 PM
  - Section L5101: Tue. Oct. 22nd, 6-8 PM
  - Section L0201: Tue. Oct. 22nd, 1-3 PM
  - Same room and time as the lecture
  - For undergraduate and graduate students

- Everything covered up to the end of today's lecture
  - Emphasis on the slides, problem set, and sample midterm provided.

- I won't have office hours next week.

Role of Transport Layer

- Link layer
  - Transfer bit frames between neighboring nodes
  - E.g., Ethernet
- Network layer
  - Logical communication between nodes
  - Hides details of the link technology
  - E.g., IP
- Transport layer
  - Communication between processes (e.g., socket)
  - Relies on network layer and serves the application layer
  - E.g., TCP and UDP
- Application layer
  - Communication for specific applications
  - E.g., HyperText Transfer Protocol (HTTP), File Transfer Protocol (FTP), Network News Transfer Protocol (NNTP)

Today's Lecture

- Principles underlying transport-layer services
  - (De)multiplexing
  - Detecting corruption
  - Reliable delivery
  - Flow control
- Transport-layer protocols in the Internet
  - User Datagram Protocol (UDP)
  - Transmission Control Protocol (TCP)

Transport Protocols

- Provide logical communication between application processes running on different hosts

- Run on end hosts
  - Sender: breaks application messages into segments, and passes to network layer
  - Receiver: reassembles segments into messages, passes to application layer

- Multiple transport protocol available to applications
  - Internet: TCP and UDP
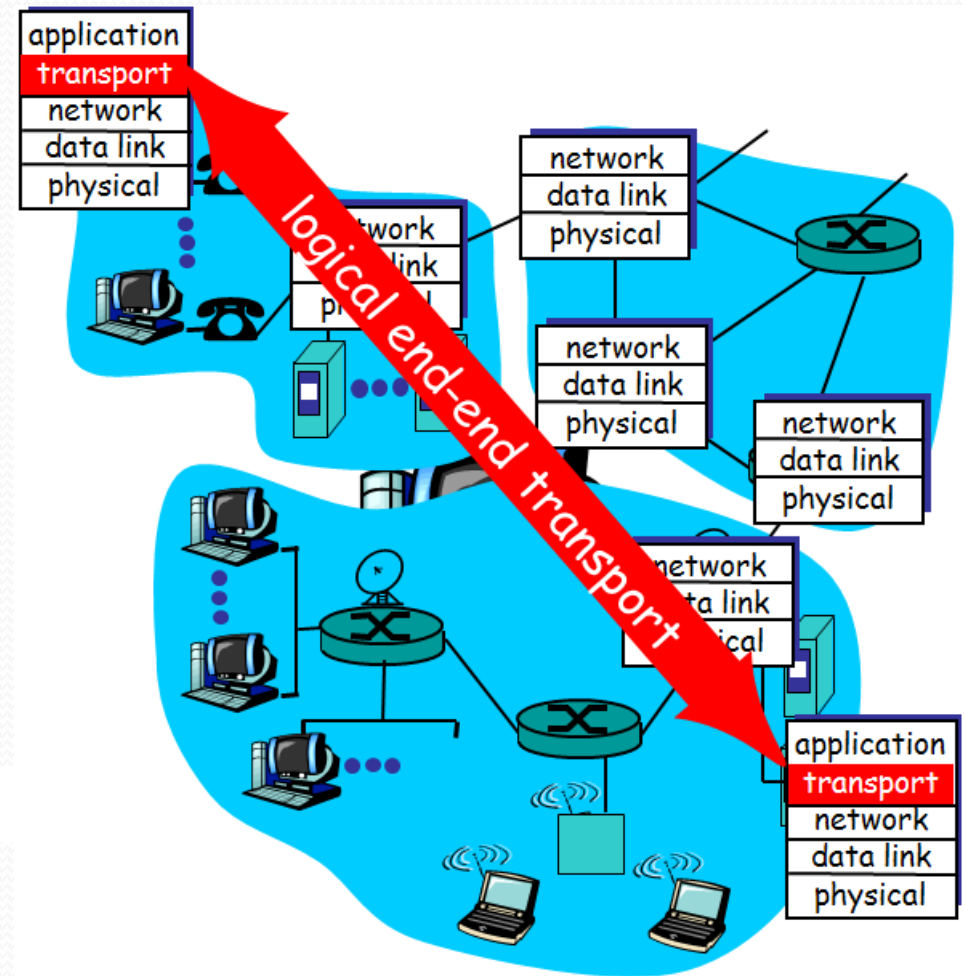
Internet Transport Protocols

- Datagram messaging service (UDP)
  - No-frills extension of "best-effort" IP
- Reliable, in-order delivery (TCP)
  - Connection set-up
  - Discarding of corrupted packets
  - Retransmission of lost packets
  - Flow control
  - Congestion control (next lecture)
- Other services not available
  - Delay guarantees
  - Bandwidth guarantees

# Multiplexing and Demultiplexing

- Host receives IP datagrams
  - Each datagram has source and destination IP address,
  - Each datagram carries one transport-layer segment
  - Each segment has source and destination port number
- Host uses IP addresses and port numbers to direct the segment to appropriate socket

32 bits

source port #	dest port #
other header fields	
application data (message)	
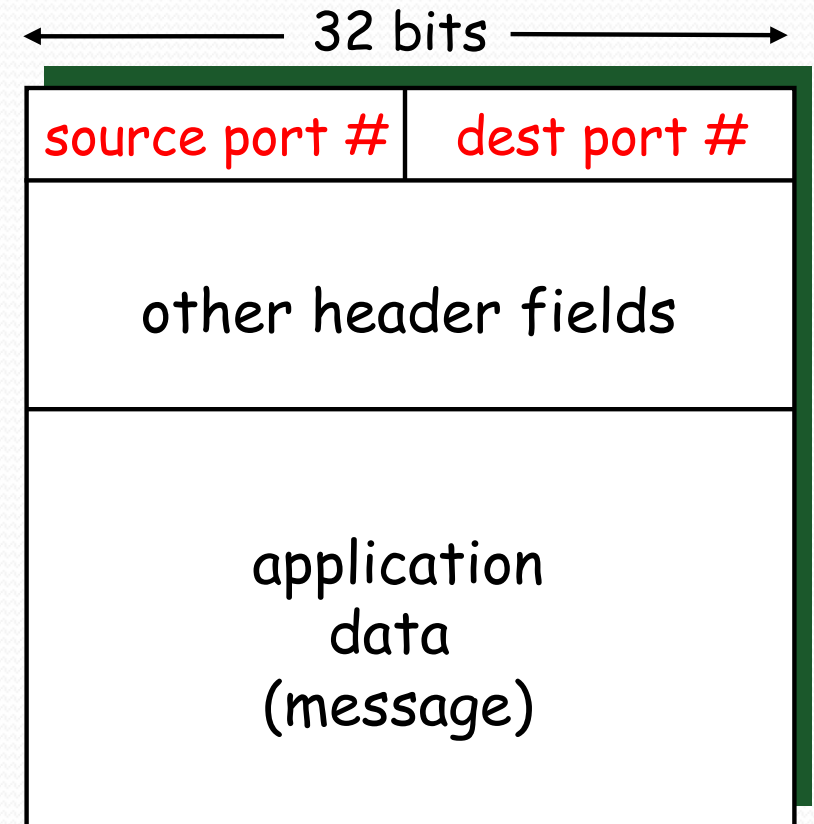
TCP/UDP segment format

Unreliable Message Delivery Service

- Lightweight communication between processes
  - Avoid overhead and delays of ordered, reliable delivery
  - Send messages to and receive them from a socket
- User Datagram Protocol (UDP)
  - IP plus port numbers to support (de)multiplexing
 - Optional error checking on the packet contents

SRC port	DST port
checksum	length
DATA	
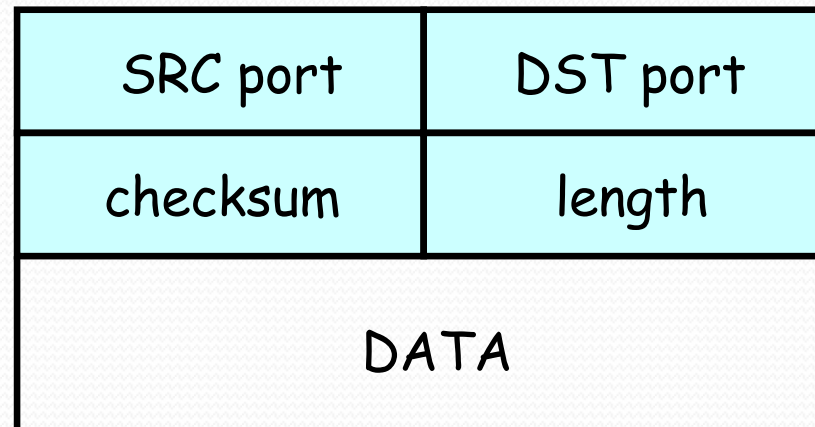
Why Would Anyone Use UDP?

- Finer control over what data is sent and when
  - As soon as an application process writes into the socket
  - … UDP will package the data and send the packet
- No delay for connection establishment
  - UDP just blasts away without any formal preliminaries
  - … which avoids introducing any unnecessary delays
- No connection state
  - No allocation of buffers, parameters, sequence #s, etc.
  - … making it easier to handle many active clients at once
- Small packet header overhead
  - UDP header is only eight-bytes long

Popular Applications That Use UDP

- Multimedia streaming
  - Retransmitting lost/corrupted packets is not worthwhile
  - By the time the packet is retransmitted, it's too late
  - E.g., telephone calls, video conferencing, gaming
- Simple query protocols like Domain Name System
  - Overhead of connection establishment is overkill
  - Easier to have application retransmit if needed

"Address for www.cnn.com?"

"12.3.4.15"
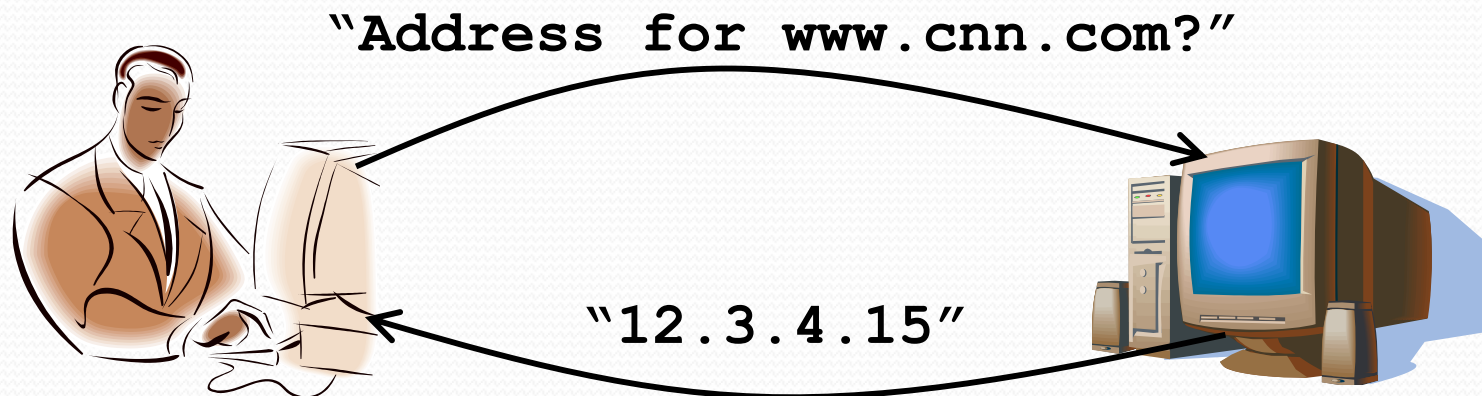
Transmission Control Protocol (TCP)

- Connection oriented
  - Explicit set-up and tear-down of TCP session
- Stream-of-bytes service
  - Sends and receives a stream of bytes, not messages
- Reliable, in-order delivery
  - Checksums to detect corrupted data
  - Acknowledgments & retransmissions for reliable delivery
  - Sequence numbers to detect losses and reorder data
- Flow control
  - Prevent overflow of the receiver's buffer space
- Congestion control
  - Adapt to network congestion for the greater good

An Analogy: Talking on a Cell Phone

- Alice and Bob on their cell phones
  - Both Alice and Bob are talking
- What if Bob couldn't understand Alice?
  - Bob asks Alice to repeat what she said
- What if Bob hasn't heard Alice for a while?
  - Is Alice just being quiet?
  - Or, have Bob and Alice lost reception?
  - How long should Bob just keep on talking?
  - Maybe Alice should periodically say "uh huh"
  - ... or Bob should ask "Can you hear me now?" ☺

Some Take-Aways from the Example

- Acknowledgments from receiver
  - Positive: "okay" or "ACK"
  - Negative: "please repeat that" or "NACK"
- Timeout by the sender ("stop and wait")
  - Don't wait indefinitely without receiving some response
  - … whether a positive or a negative acknowledgment
- Retransmission by the sender
  - After receiving a "NACK" from the receiver
  - After receiving no feedback from the receiver

Challenges of Reliable Data Transfer

- Over a perfectly reliable channel
  - All of the data arrives in order, just as it was sent
  - Simple: sender sends data, and receiver receives data
- Over a channel with bit errors
  - All of the data arrives in order, but some bits corrupted
  - Receiver detects errors and says "please repeat that"
  - Sender retransmits the data that were corrupted
- Over a lossy channel with bit errors
  - Some data are missing, and some bits are corrupted
  - Receiver detects errors but cannot always detect loss
  - Sender must wait for acknowledgment ("ACK" or "OK")
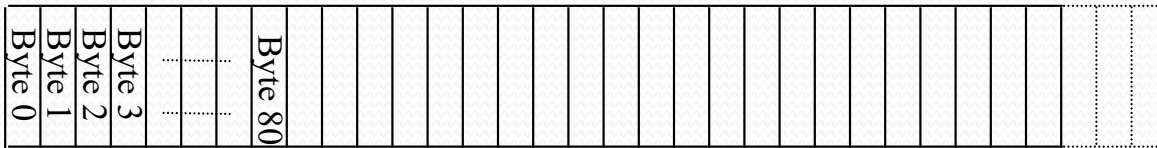 - … and retransmit data after some time if no ACK arrives

TCP Support for Reliable Delivery

- Checksum
  - Used to detect corrupted data at the receiver
  - …leading the receiver to drop the packet
- Sequence numbers
  - Used to detect missing data
  - … and for putting the data back in order
- Retransmission
  - Sender retransmits lost or corrupted data
  - Timeout based on estimates of round-trip time
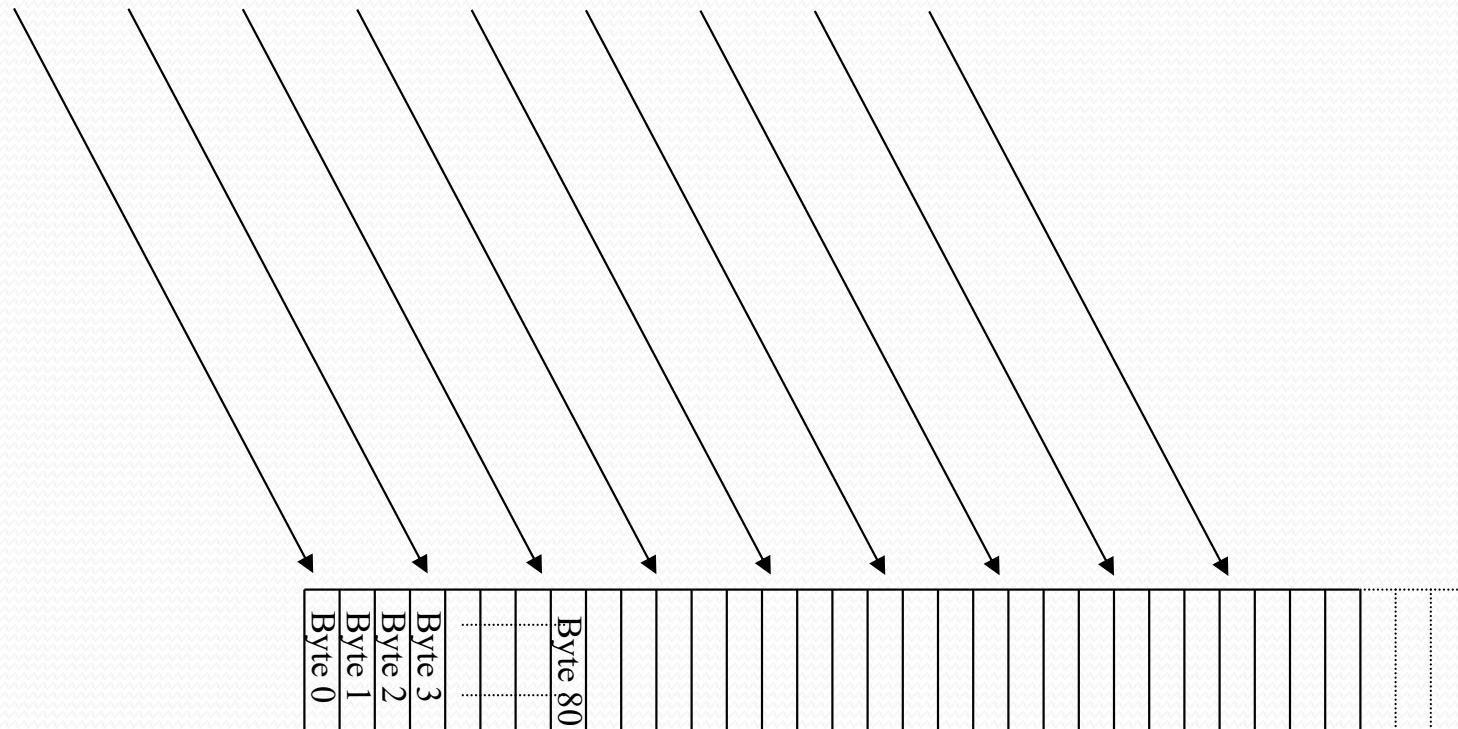 - Fast retransmit algorithm for rapid retransmission

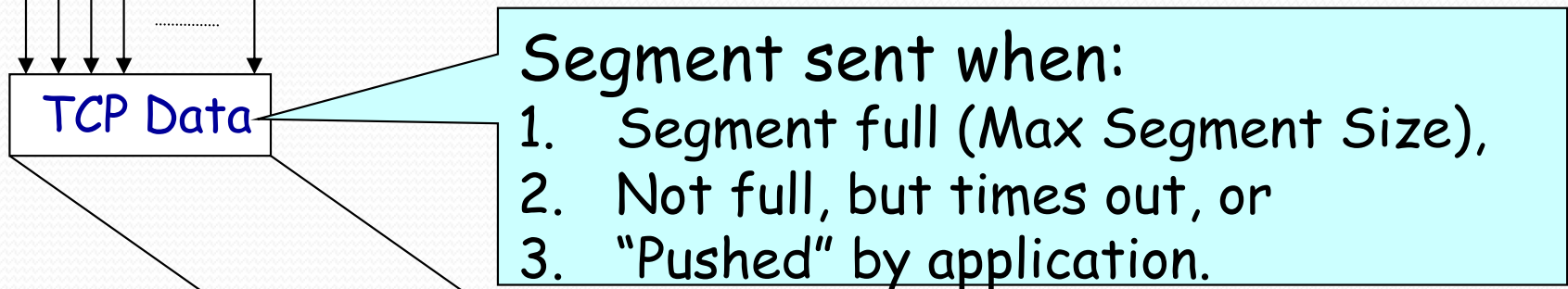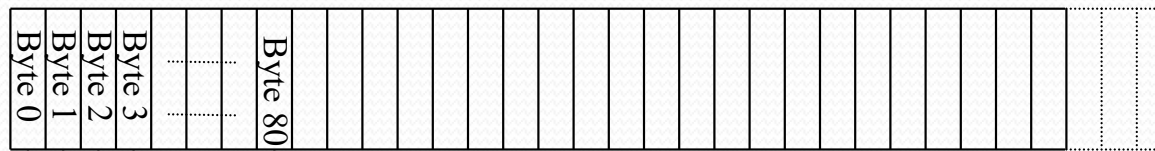# TCP Segments
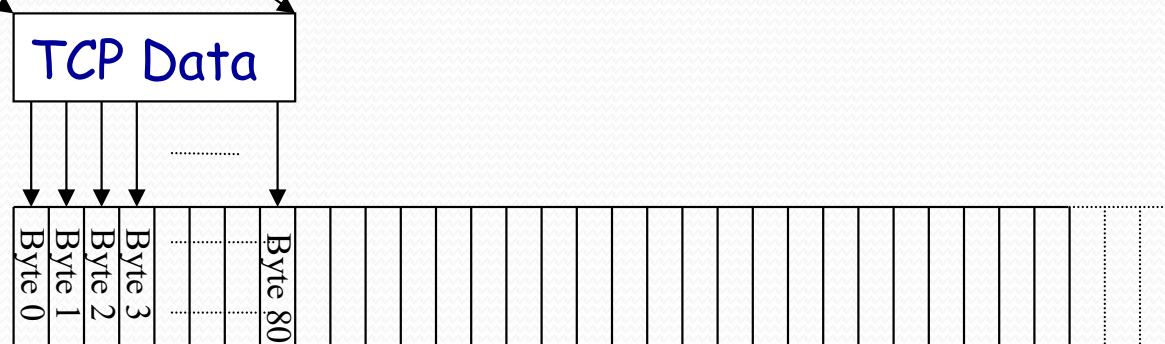
TCP "Stream of Bytes" Service

Host A

Host B

…Emulated Using TCP "Segments"

Host A

Host B

| Byte 0 | Byte 1 | Byte 2 | Byte 3 | ……… | Byte 80 |

TCP Data

Segment sent when:
1. Segment full (Max Segment Size),
2. Not full, but times out, or
3. "Pushed" by application.

TCP Data

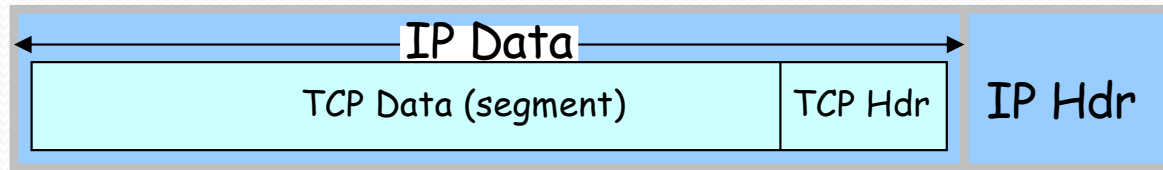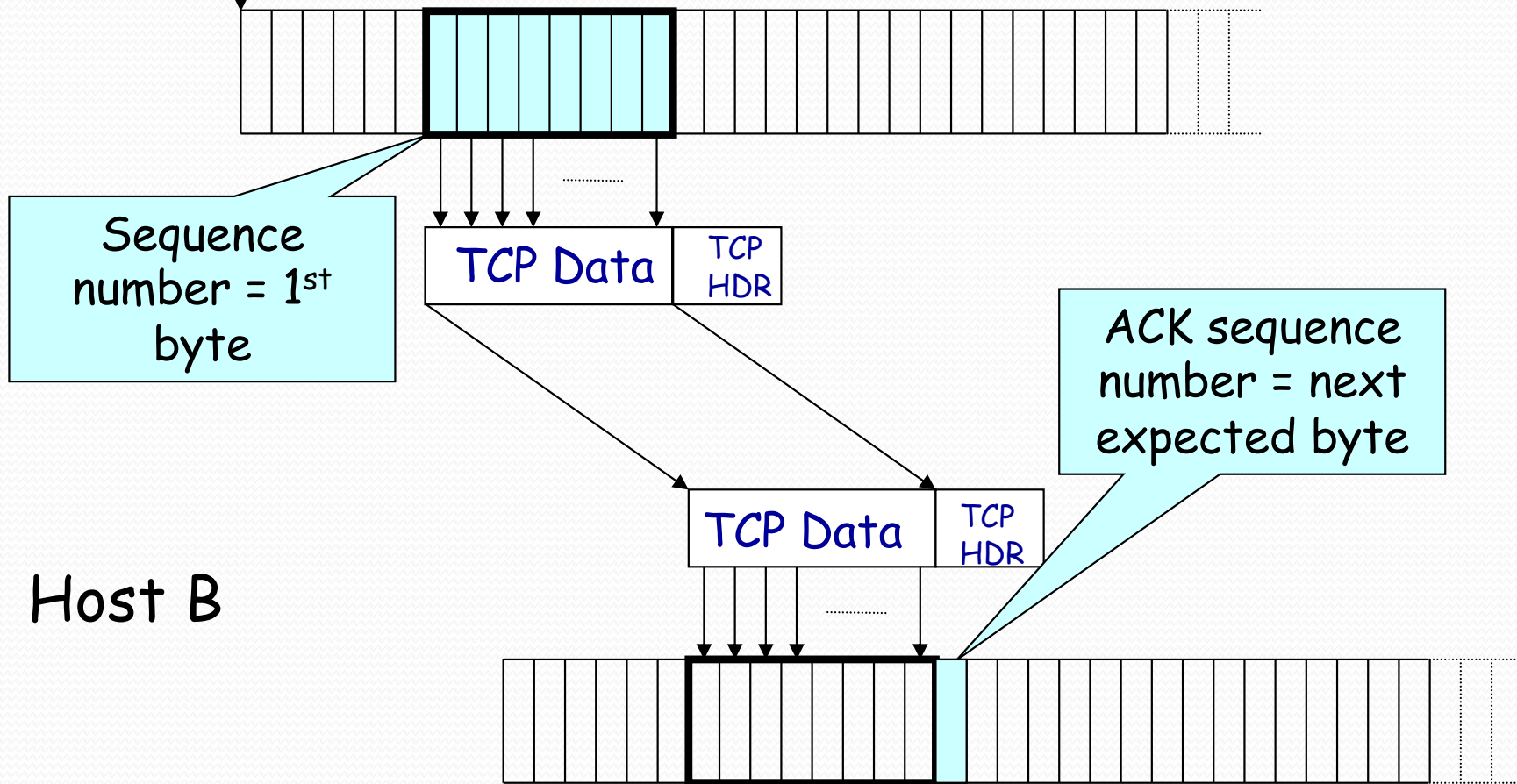| Byte 0 | Byte 1 | Byte 2 | Byte 3 | ……… | Byte 80 |

TCP Segment

- IP packet
  - No bigger than Maximum Transmission Unit (MTU)
  - E.g., up to 1500 bytes on an Ethernet
- TCP packet
  - IP packet with a TCP header and data inside
  - TCP header is typically 20 bytes long
- TCP segment
  - No more than Maximum Segment Size (MSS) bytes
 - E.g., up to 1460 consecutive bytes from the stream

Sequence Numbers

Initial Sequence Number (ISN)

- Sequence number for the very first byte
  - E.g., Why not a de facto ISN of 0?
- Practical issue
  - IP addresses and port #s uniquely identify a connection
  - Eventually, though, these port #s do get used again
  - … and there is a chance an old packet is still in flight
  - … and might be associated with the new connection
- So, TCP requires changing the ISN over time
  - Set from a 32-bit clock that ticks every 4 microseconds
  - … which only wraps around once every 4.55 hours!
- But, this means the hosts need to exchange ISNs

TCP Three-Way Handshake
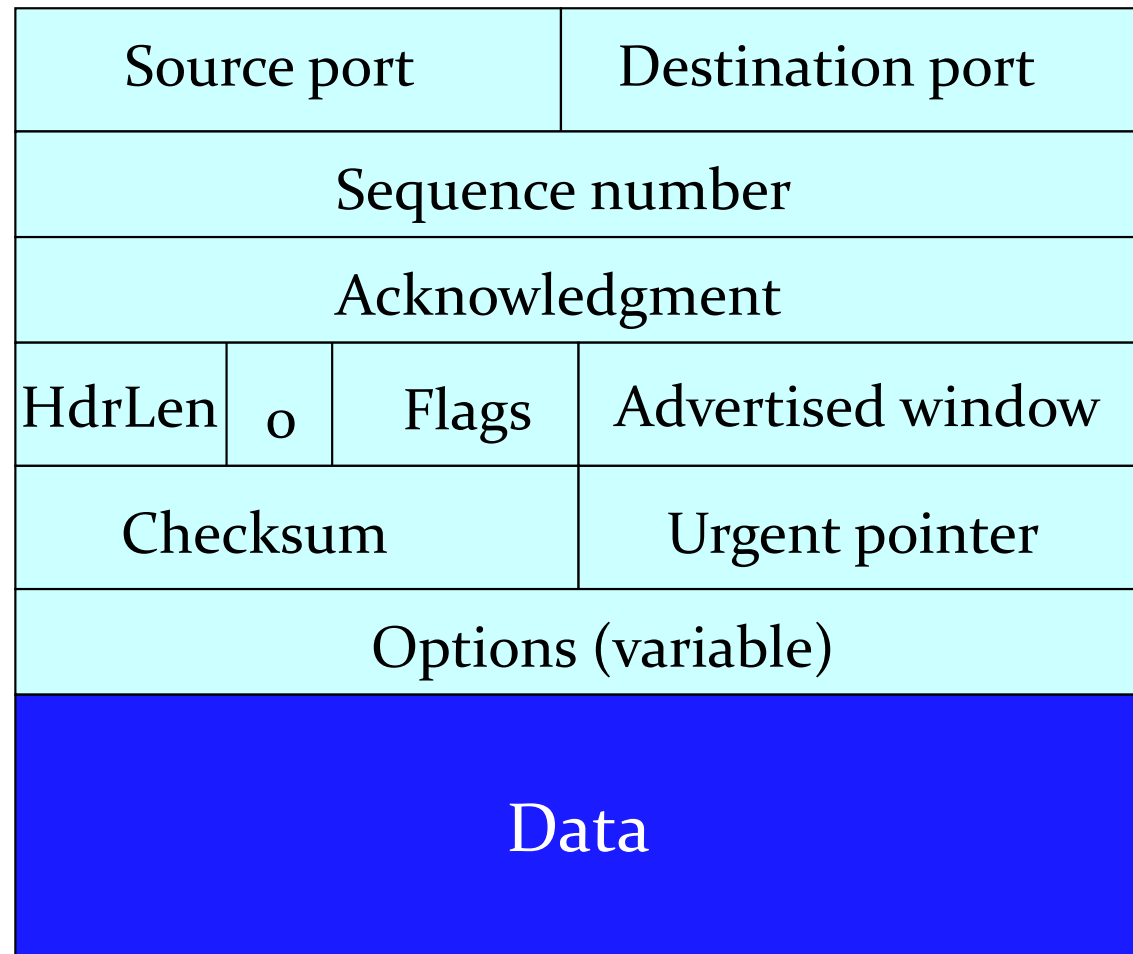
Establishing a TCP Connection
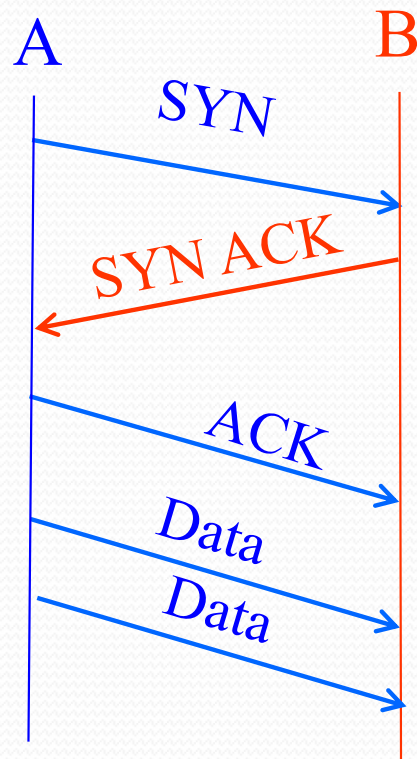
Each host tells its ISN to the other host.

- Three-way handshake to establish connection
  - Host A sends a **SYN** (open) to the host B
  - Host B returns a SYN acknowledgment (**SYN ACK**)
  - Host A sends an **ACK** to acknowledge the SYN ACK

TCP Header

Flags: SYN
FIN
RST
PSH
URG
ACK

Source port		Destination port	
Sequence number			
Acknowledgment			
HdrLen	0	Flags	Advertised window
Checksum		Urgent pointer	
Options (variable)			
Data			

Step 1: A's Initial SYN Packet

Flags: SYN
         FIN
         RST
         PSH
         URG
         ACK

A's port		B's port	
A's Initial Sequence Number			
Acknowledgment			
20	0	Flags	Advertised window
Checksum		Urgent pointer	
Options (variable)			

A tells B it wants to open a connection…

Step 2: B's SYN-ACK Packet

Flags: SYN
         FIN
         RST
         PSH
         URG
         ACK

B's port	A's port
B's Initial Sequence Number	
A's ISN plus 1	

20	0	Flags	Advertised window
Checksum			Urgent pointer
Options (variable)			

B tells A it accepts, and is ready to hear the next byte…

… upon receiving this packet, A can start sending data

Step 3: A's ACK of the SYN-ACK

Flags: SYN
FIN
RST
PSH
URG
ACK

A's port				B's port	
Sequence number					
B's ISN plus 1					
20	0	Flags		Advertised window	
Checksum			Urgent pointer		
Options (variable)					

A tells B it is okay to start sending

… upon receiving this packet, B can start sending data

What if the SYN Packet Gets Lost?

- Suppose the SYN packet gets lost
  - Packet is lost inside the network, or
  - Server rejects the packet (e.g., listen queue is full)
- Eventually, no SYN-ACK arrives
  - Sender sets a timer and wait for the SYN-ACK
  - … and retransmits the SYN if needed
- How should the TCP sender set the timer?
  - Sender has no idea how far away the receiver is
  - Hard to guess a reasonable length of time to wait
 - Some TCPs use a default of 3 or 6 seconds

SYN Loss and Web Downloads

- User clicks on a hypertext link
  - Browser creates a socket and does a "connect"
  - The "connect" triggers the OS to transmit a SYN

- If the SYN is lost...
  - The 3-6 seconds of delay may be very long
  - The user may get impatient
  - ... and click the hyperlink again, or click "reload"

- User triggers an "abort" of the "connect"
  - Browser creates a new socket and does a "connect"
  - Essentially, forces a faster send of a new SYN packet!
 - Sometimes very effective, and the page comes fast

TCP Retransmissions

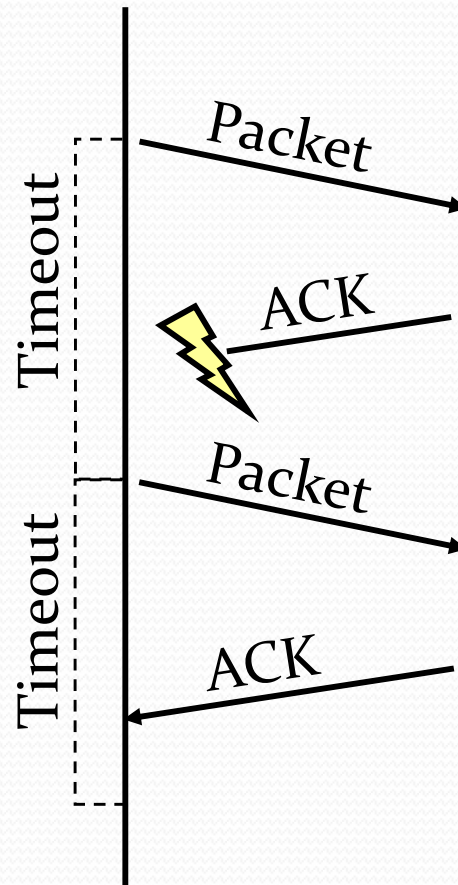# Automatic Repeat reQuest (ARQ)

- Automatic Repeat reQuest
  - Receiver sends acknowledgment (ACK) when it receives packet
  - Sender waits for ACK and timeouts if it does not arrive within some time period
- Simplest ARQ protocol
  - Stop and wait
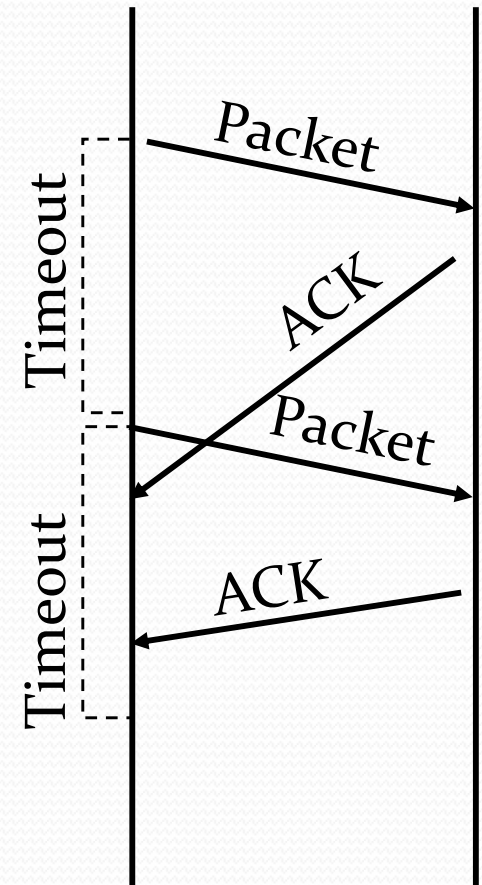 - Send a packet, stop and wait until ACK arrives
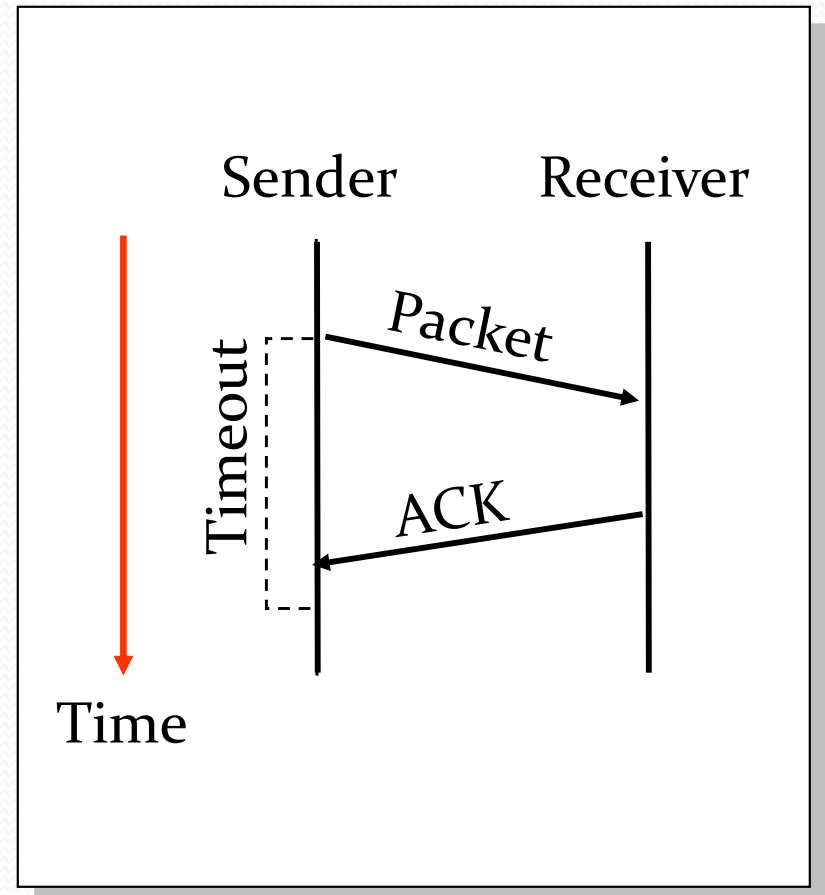
Reasons for Retransmission
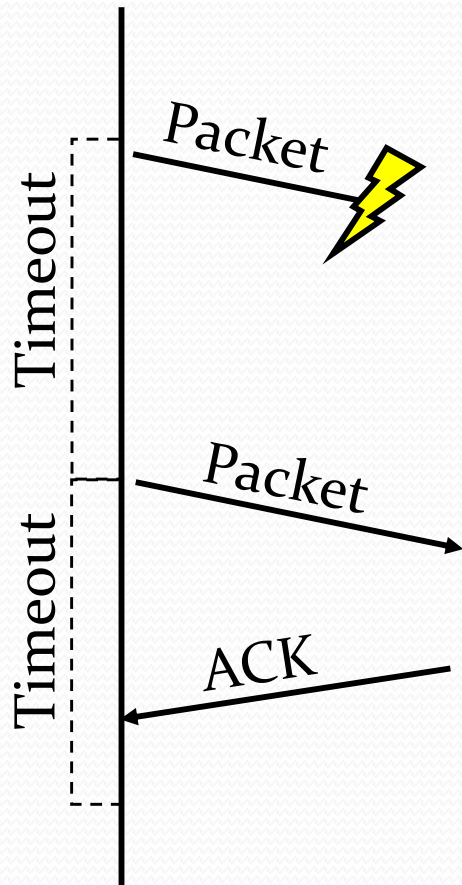
Packet lost

**ACK lost**
DUPLICATE PACKET

Early timeout
DUPLICATE PACKETS

How Long Should Sender Wait?

- Sender sets a timeout to wait for an ACK
  - Too short: wasted retransmissions
  - Too long: excessive delays when packet lost
- TCP sets timeout as a function of the RTT
  - Expect ACK to arrive after an RTT
  - … plus a fudge factor to account for queuing
- But, how does the sender know the RTT?
  - Can estimate the RTT by watching the ACKs
  - Smooth estimate: keep a running average of the RTT
    - EstimatedRTT = a * EstimatedRTT + (1 –a) * SampleRTT
  - Compute timeout: TimeOut = 2 * EstimatedRTT

Example RTT Estimation

RTT: gaia.cs.umass.edu to fantasia.eurecom.fr
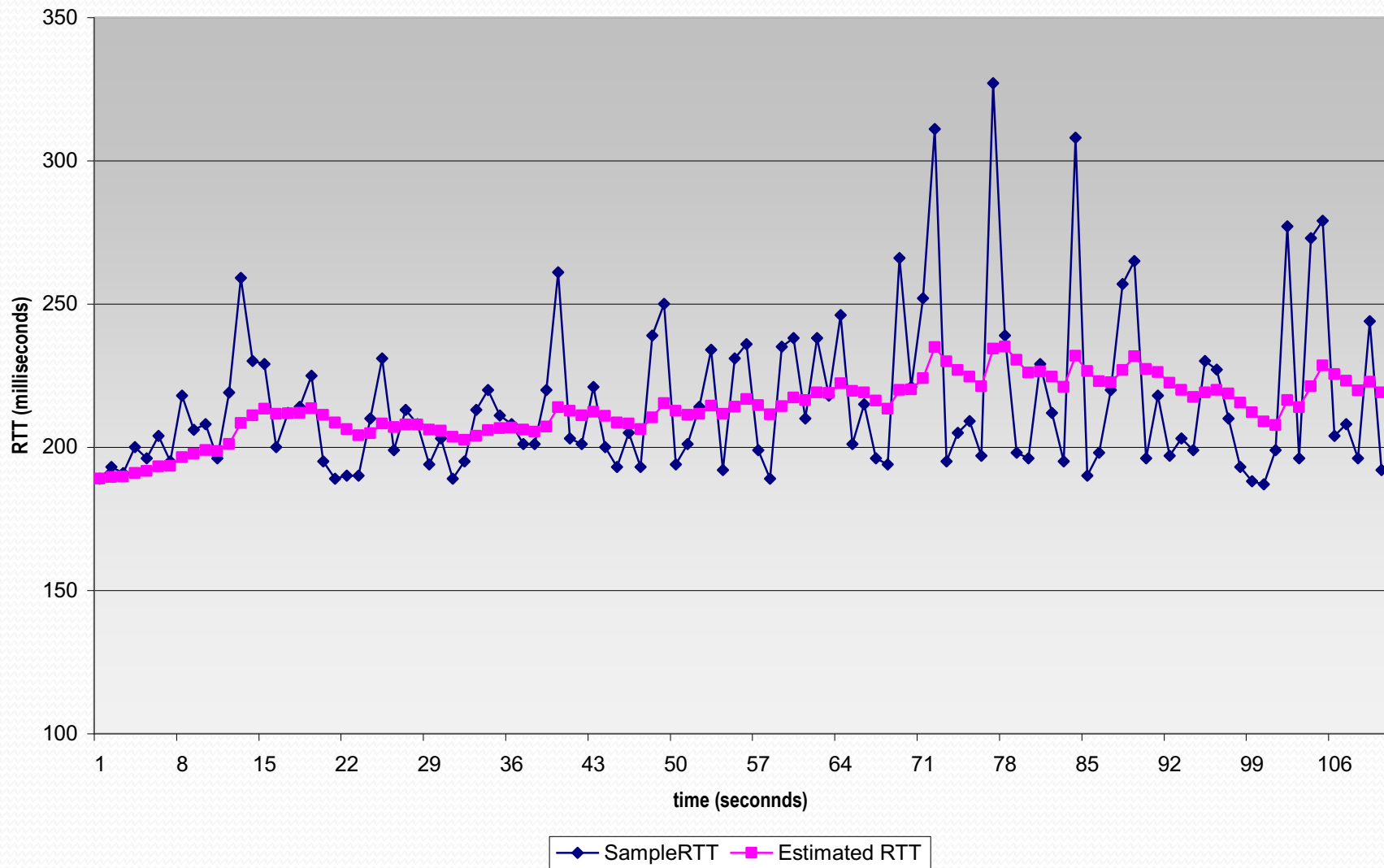
A Flaw in This Approach

- An ACK doesn't really acknowledge a transmission
  - Rather, it acknowledges receipt of the data
- Consider a retransmission of a lost packet
  - If you assume the ACK goes with the 1st transmission
  - … the SampleRTT comes out way too large
- Consider a duplicate packet
  - If you assume the ACK goes with the 2nd transmission
  - … the Sample RTT comes out way too small
- Simple solution in the Karn/Partridge algorithm
  - Only collect samples for segments sent one single time
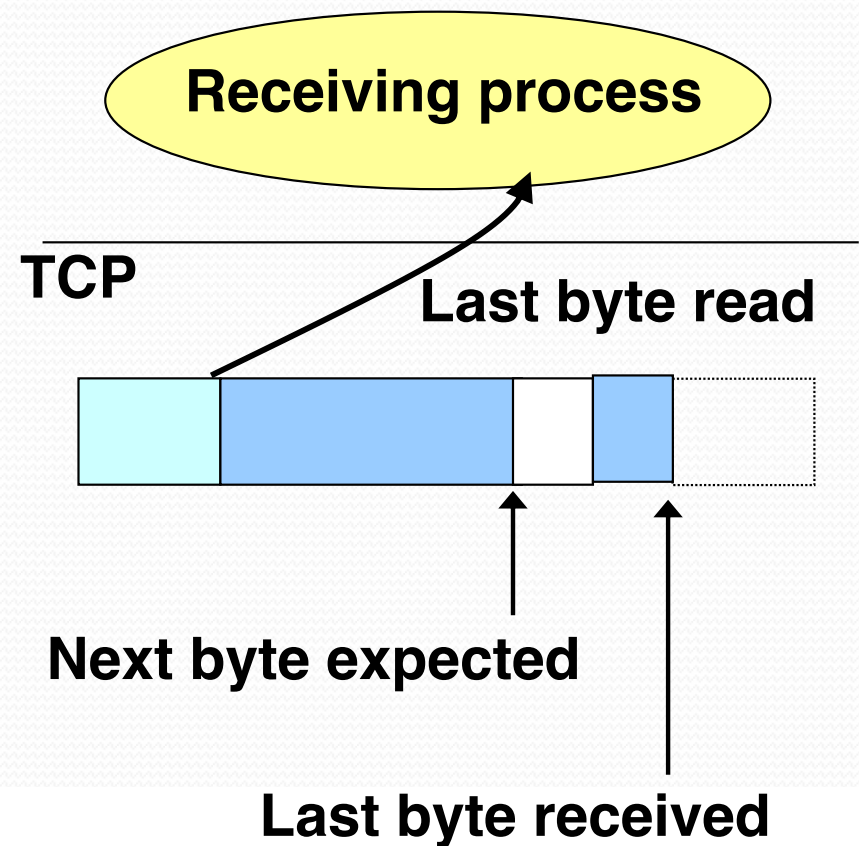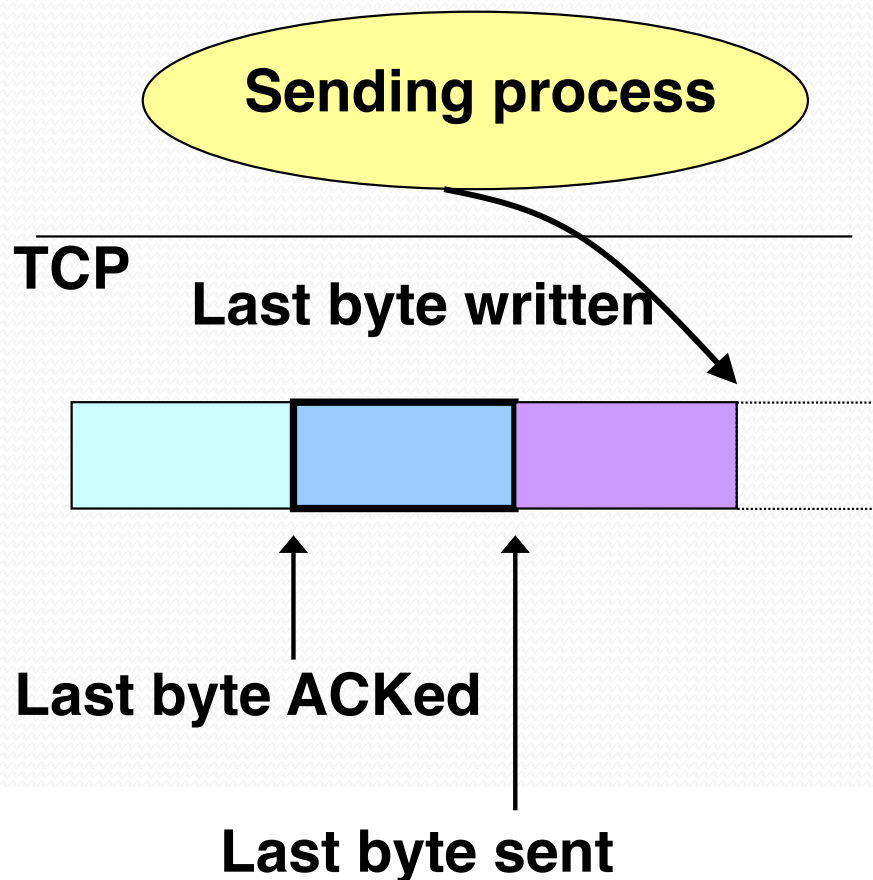
TCP Sliding Window

Motivation for Sliding Window

- Stop-and-wait is inefficient
  - Only one TCP segment is "in flight" at a time
  - Especially bad when delay-bandwidth product is high
- Numerical example
  - 1.5 Mbps link with a 45 msec round-trip time (RTT)
    - Delay-bandwidth product is 67.5 Kbits (or 8 KBytes)
  - But, sender can send at most one packet per RTT
    - Assuming a segment size of 1 KB (8 Kbits)
    - ... leads to 8 Kbits/segment / 45 msec/segment ➔ 182 Kbps
   - That's just one-eighth of the 1.5 Mbps link capacity
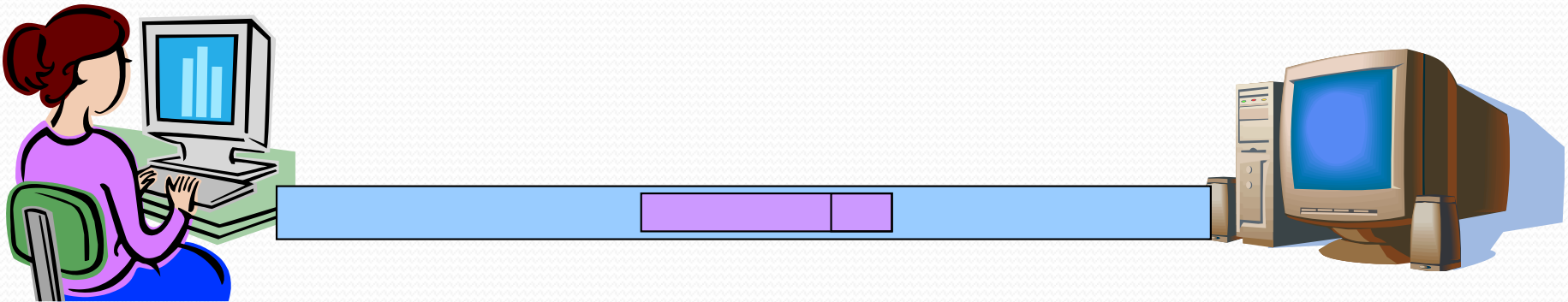
Sliding Window

- Allow a larger amount of data "in flight"
  - Allow sender to get ahead of the receiver
  - … though not too far ahead

Sending process

TCP

Last byte written

Last byte ACKed

Last byte sent

Receiving process

TCP

Last byte read

Next byte expected

Last byte received

Receiver Buffering

- Window size
  - Amount that can be sent without acknowledgment
  - Receiver needs to be able to store this amount of data
- Receiver advertises the window to the sender
  - Tells the sender the amount of free space left
  - ... and the sender agrees not to exceed this amount

Window Size

Data ACK'd | Outstanding Un-ack'd data | Data OK to send | Data not OK to send yet
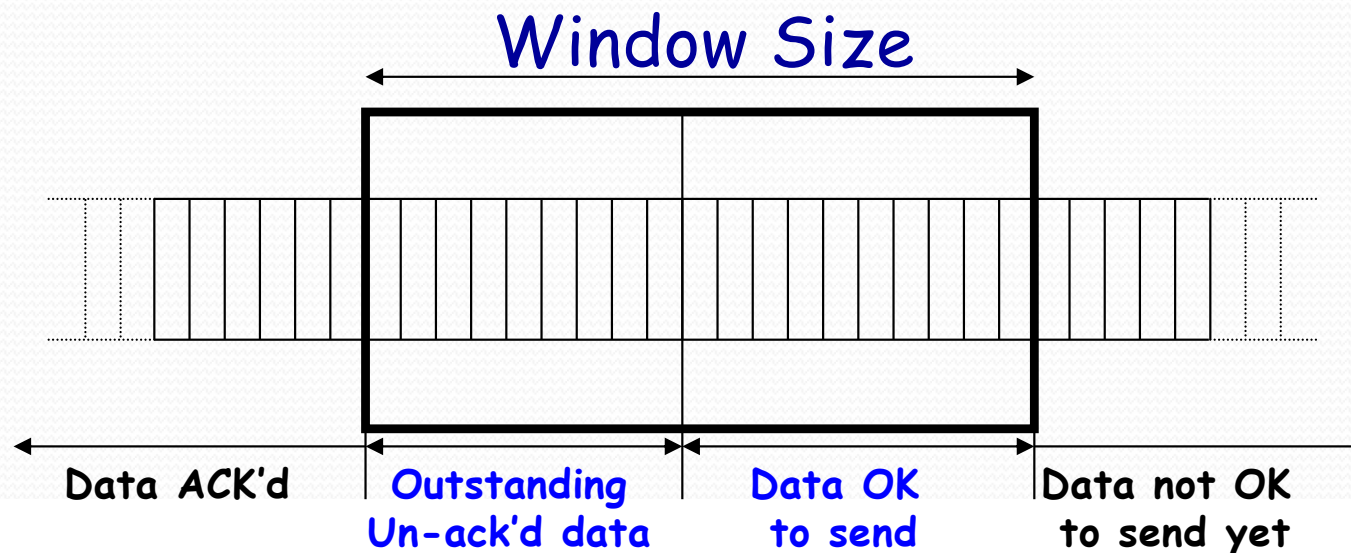
TCP Header for Receiver Buffering

Flags: SYN
         FIN
         RST
         PSH
         URG
         ACK

Source port			Destination port
Sequence number			
Acknowledgment			
HdrLen	0	Flags	**Advertised window**
Checksum			Urgent pointer
Options (variable)			
Data			

Fast Retransmission

Timeout is Inefficient

- Timeout-based retransmission
  - Sender transmits a packet and waits until timer expires
  - … and then retransmits from the lost packet onward
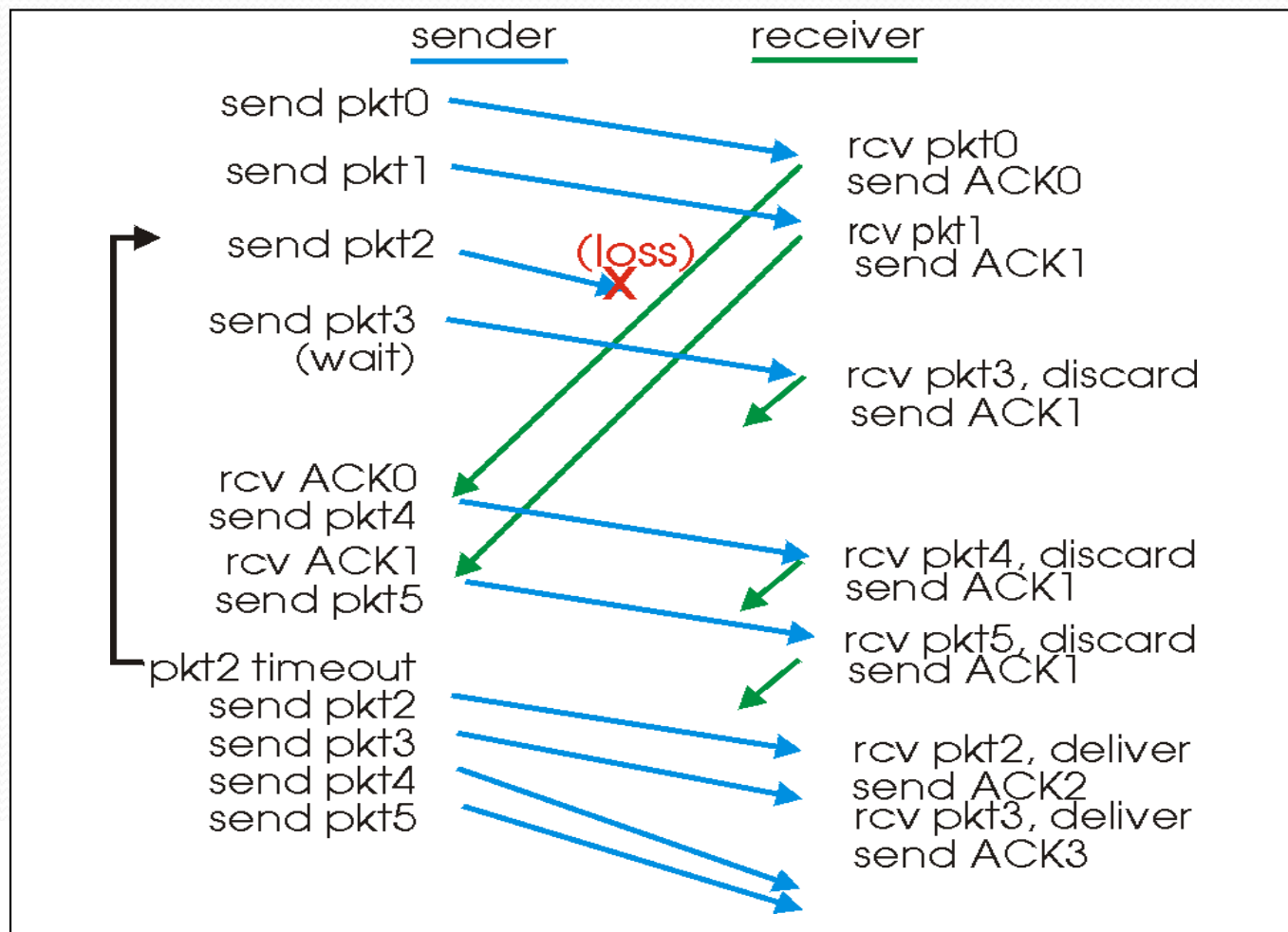
Fast Retransmission

- Better solution possible under sliding window
  - Although packet n might have been lost
  - … packets n+1, n+2, and so on might get through
- Idea: have the receiver send ACK packets
  - ACK says that receiver is still awaiting nth packet
    - And repeated ACKs suggest later packets have arrived
  - Sender can view the "duplicate ACKs" as an early hint
    - … that the nth packet must have been lost
    - … and perform the retransmission early
- Fast retransmission
  - Sender retransmits data after the triple duplicate ACK

Effectiveness of Fast Retransmit

- When does Fast Retransmit work best?
  - Long data transfers
    - High likelihood of many packets in flight
  - High window size
    - High likelihood of many packets in flight
  - Low burstiness in packet losses
    - Higher likelihood that later packets arrive successfully
- Implications for Web traffic
  - Most Web transfers are short (e.g., 10 packets)
    - Short HTML files or small images
  - So, often there aren't many packets in flight
  - … making fast retransmit less likely to "kick in"
  - Forcing users to like "reload" more often… ☺

Tearing Down the Connection

Tearing Down the Connection
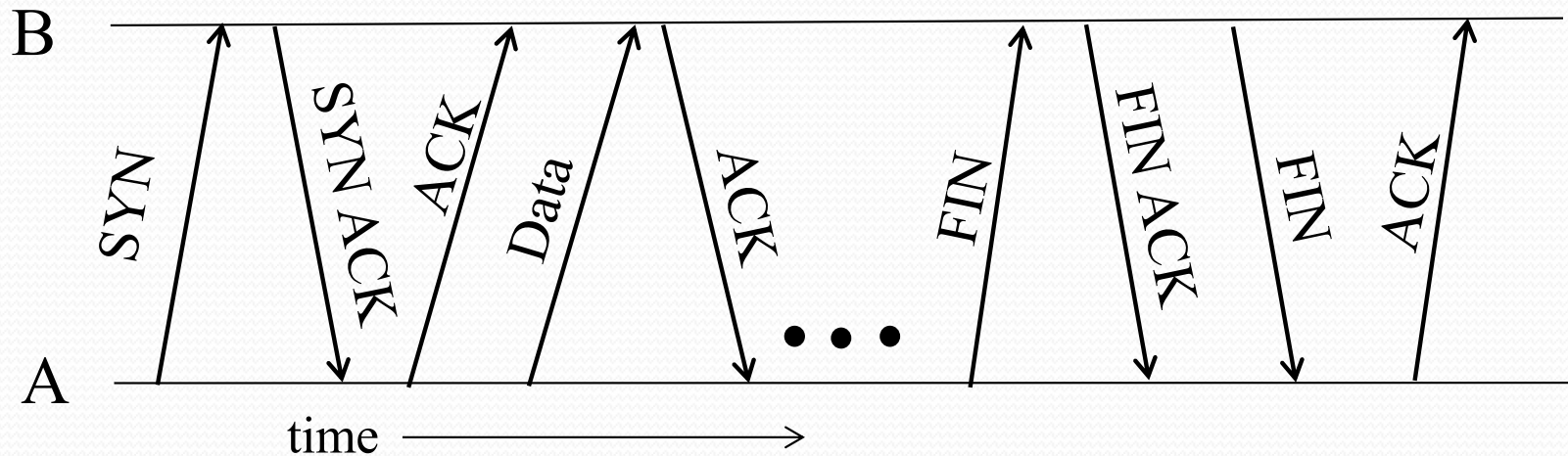
- Closing the connection
  - Finish (FIN) to close and receive remaining bytes
  - And other host sends a FIN ACK to acknowledge
  - Reset (RST) to close and not receive remaining bytes

Sending/Receiving the FIN Packet

- Sending a FIN: close()

  - Process is done sending data via the socket

  - Process invokes "close()" to close the socket

  - Once TCP has sent all of the outstanding bytes…

  - … then TCP sends a FIN

- Receiving a FIN: EOF

  - Process is reading data from the socket

  - Eventually, the attempt to read returns an EOF

Conclusions

- Transport protocols
  - Multiplexing and demultiplexing
  - Sequence numbers
  - Window-based flow control
  - Timer-based retransmission
  - Checksum-based error detection

- Next lecture (after midterm)
  - Congestion control