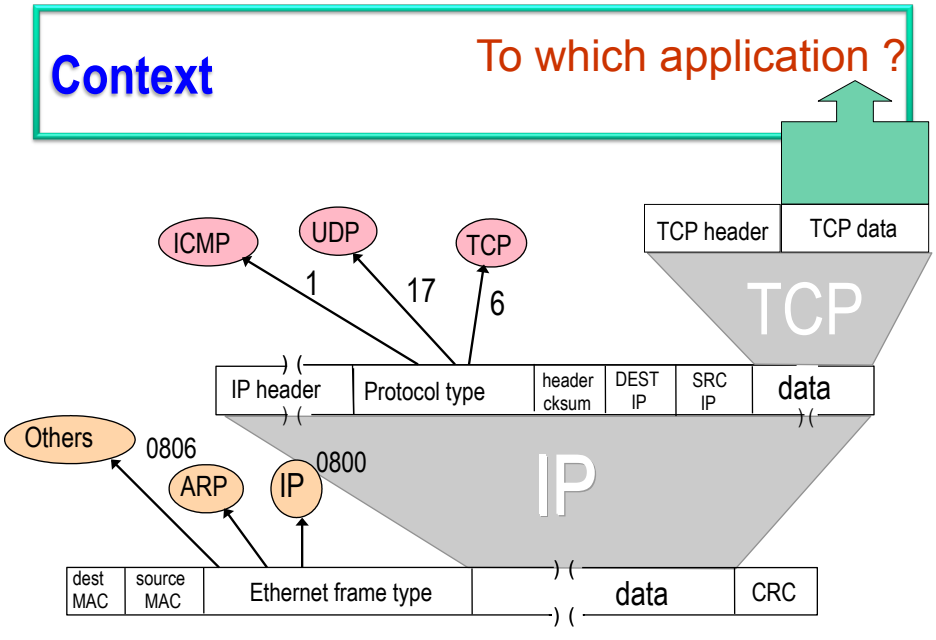
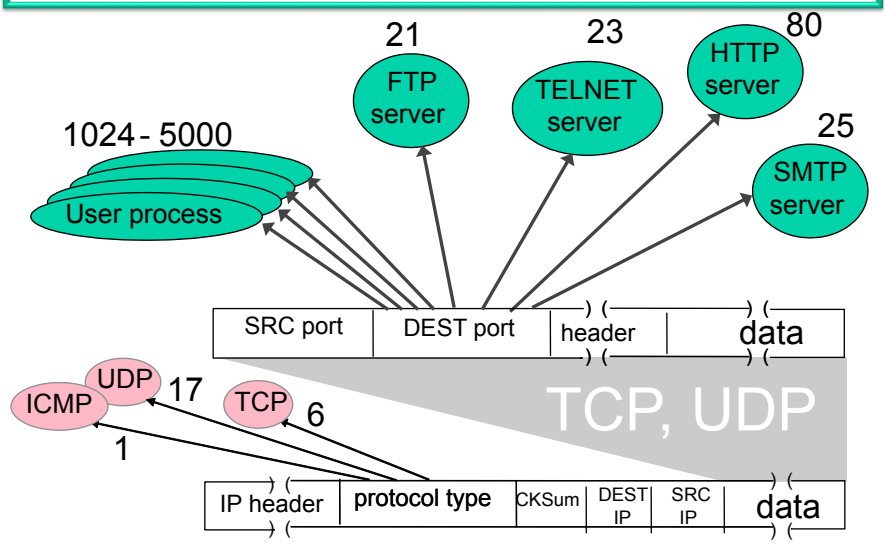
Chapters 14 – 16

Transport Layer Protocols



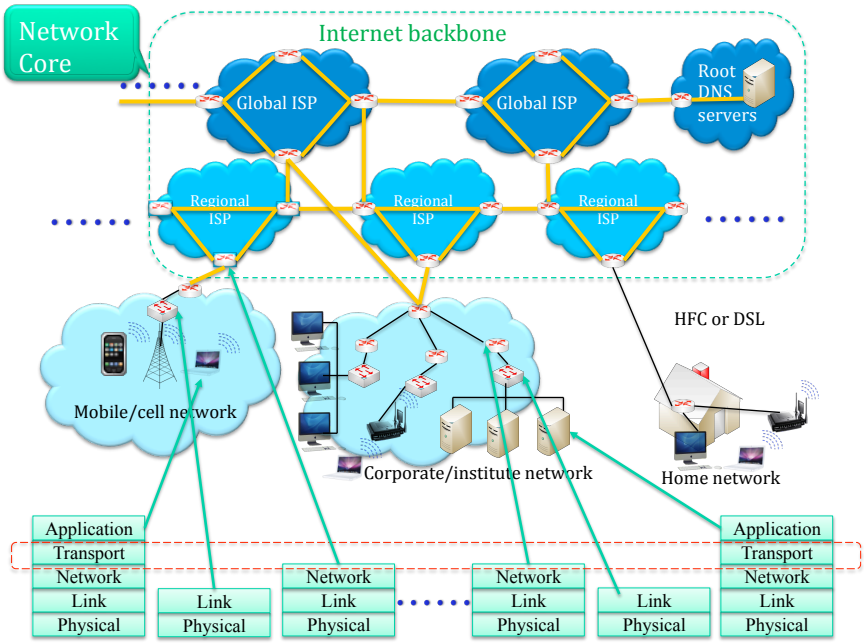
Demultiplexing – port numbers



Outline

Part 4

- Overview
  - TCP, STCP and UDP
- TCP transport protocol
  - Reliable delivery
  - Flow control
  - TCP handshake
  - Congestion control



The transport layer provides inter process communication between applications running on two different end hosts

Transport layer protocols

- Choose TCP, UDP or SCTP based on applications
  - TCP: Transmission Control Protocol
    - Reliable delivery of web, bank transactions, email messages
    - Connection setup and teardown
    - Flow control and congestion control
    - Preserving byte order
  - SCTP: Stream Control Transmission Protocol
    - Possesses all the advantages of TCP
    - Parallel delivery of multiple objects for a transaction
    - Improves security (denial of service protection) and reliability
  - UDP: User Datagram Protocol
    - Unreliable delivery
    - Connectionless
    - No order of messages
    - Primarily used for voice/audio/video

TCP vs. SCTP

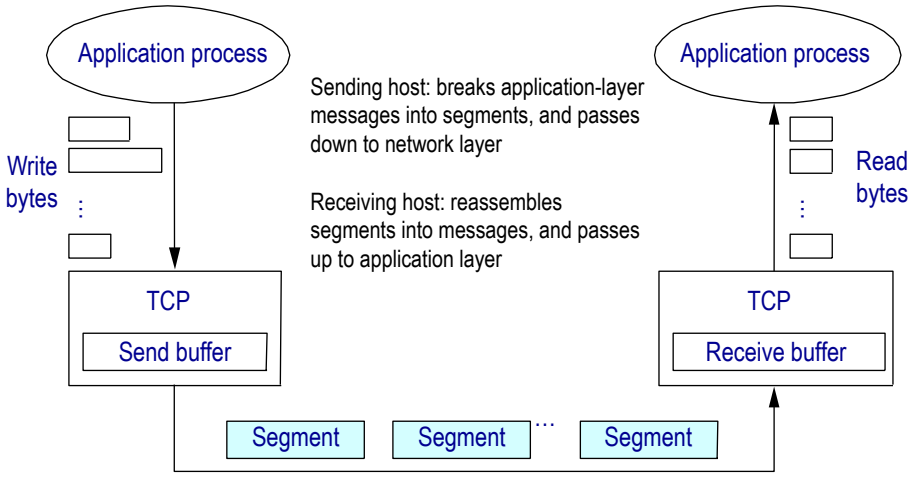
- Use http to send a home page containing multiple objects as an example:
  - If the base file of http has a lost segment, then no display of the page will occur until retransmission
  - If one of the image files has a segment lost, then the following objects cannot be sent until this image file is retransmitted correctly (Non-Persistent HTTP, serial TCP)
- Use SCTP to replace TCP for http:
  - Multiple objects are sent in parallel
  - When there is an error in some objects, the correctly received objects can be displayed first
  - Then, the retransmitted object will appear

Outline

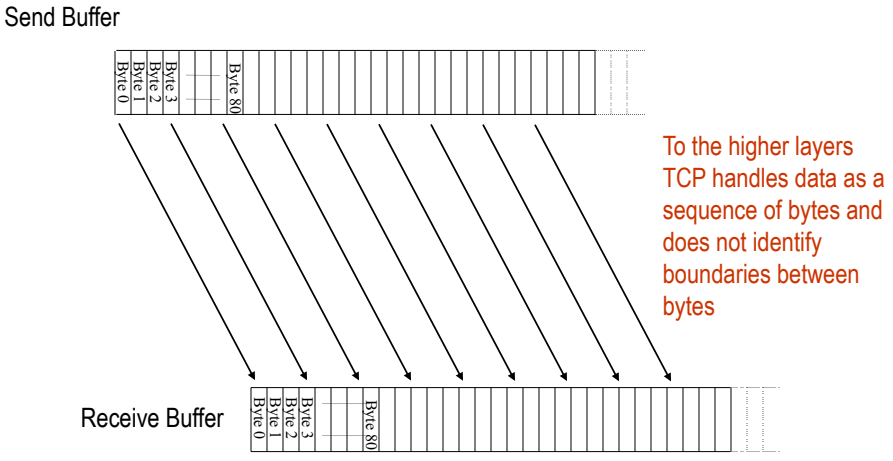
Part 4

- Overview
  - TCP, STCP and UDP
- TCP transport protocol
  - Reliable delivery
  - Flow control
  - TCP handshake
  - Congestion control

How TCP Manages a Byte Stream



TCP "Stream of Bytes" Service

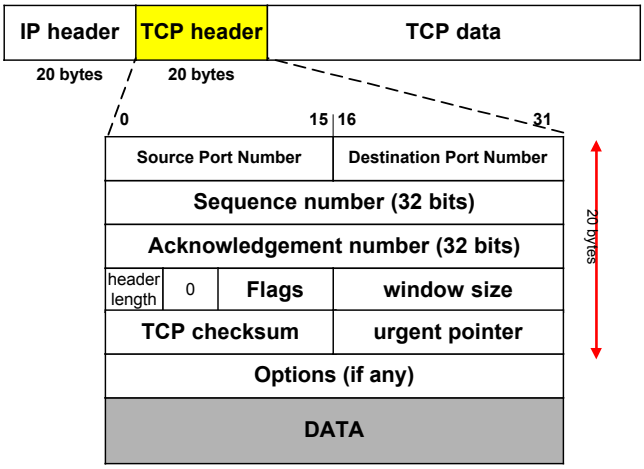


TCP Segment

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

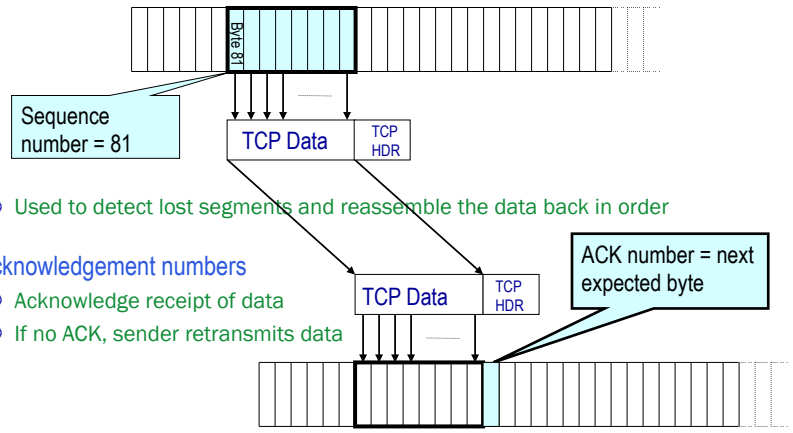
TCP Header Format

TCP segments have 20 byte header with >= 0 bytes of data.



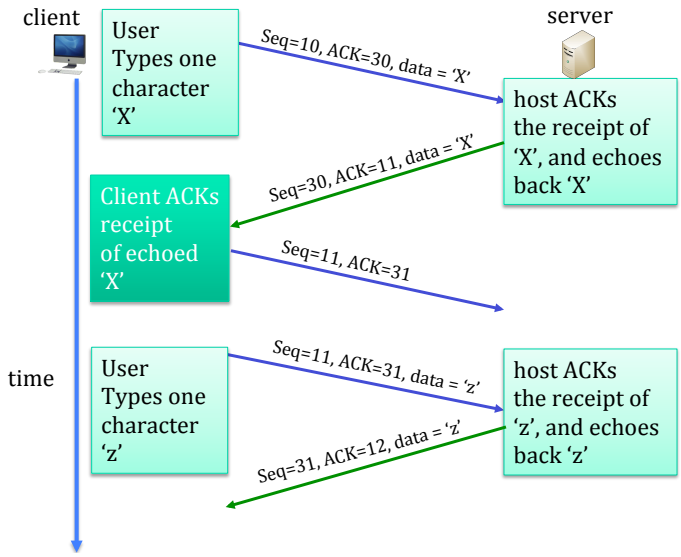
Reliable delivery: SEQ and ACK numbers

- Sequence numbers
  - Identify a byte in the byte stream

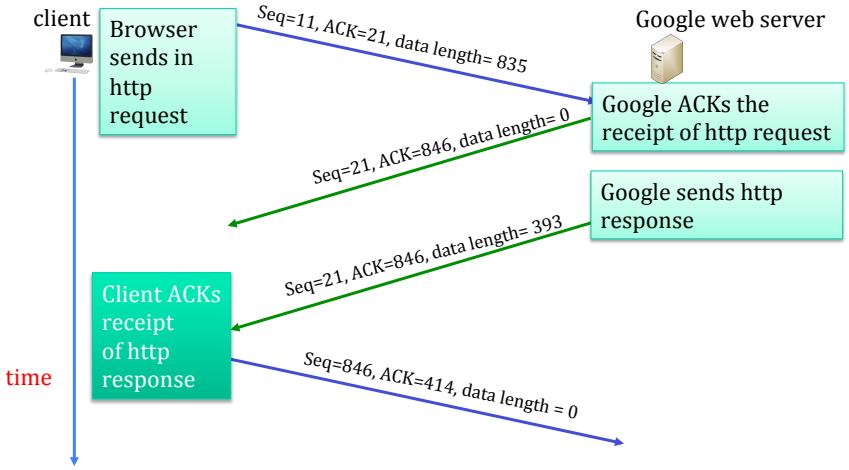


  - Used to detect lost segments and reassemble the data back in order
- Acknowledgement numbers
  - Acknowledge receipt of data
  - If no ACK, sender retransmits data

Example: SEQ and ACK in telnet



Example: SEQ and ACK in http

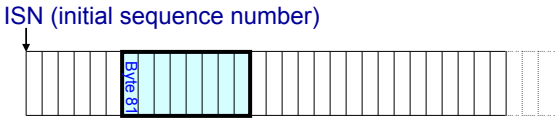


Example

- Suppose a TCP connection is transferring a file of 5000 bytes. The first byte is numbered 501. What are the SEQ and ACK numbers for each segment if data is sent in five segments, each carrying 1000 bytes?

Segment	SEQ Number (sent)	ACK Number (received)
1st		
2nd		
3rd		
4th		
5th		

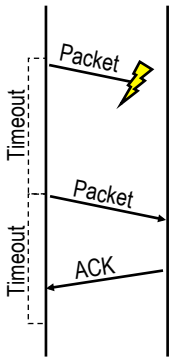
Initial Sequence Number (ISN)

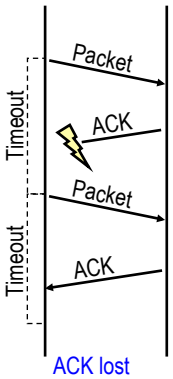


- Initial sequence number (ISN) for the very first byte of a connection
  - TCP sequence numbers are 32-bit integers in the circular range of 0 to 4,294,967,295
  - The hosts at both ends of a TCP connection exchange an Initial Sequence Number (ISN) selected at random as part of the setup of a new TCP connection
  - After the session is established, the sequence number is regularly augmented by the number of octets transferred, and transmitted to the other host
- If the initial sequence number is not chosen randomly, then it is possible to mix up packets from different connections

Detecting packet loss/delay

- ACK is used to update sender so that new packet can be sent
- Timeout based on estimated RTT (Round Trip Time) is used to determine if packet was delivered
  - Sender retransmits a packet after a timeout in waiting for ACK
- Sender is in the mode of send-stop-wait-send-next cycle
  - The easiest way for reliable transport



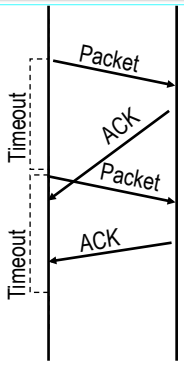


How long should the sender wait?

- 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 "round-trip time"
  - ... plus a 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

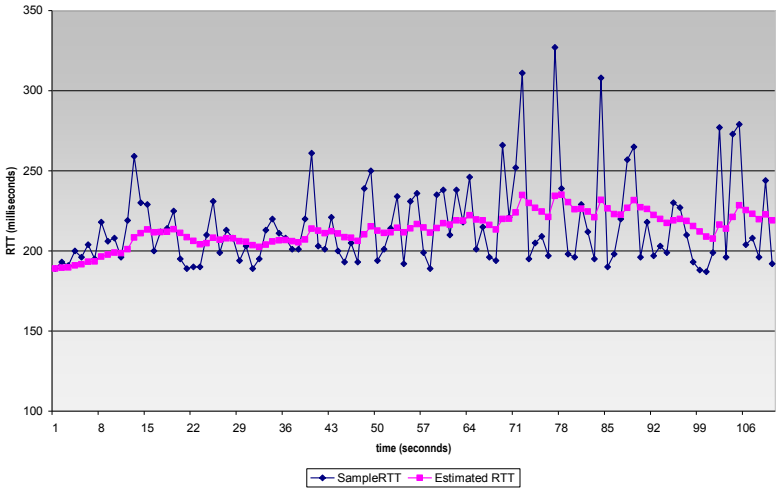
EstimatedRTT = α * EstimatedRTT + (1 –α) * SampleRTT

  - Compute timeout: TimeOut = 2 * EstimatedRTT



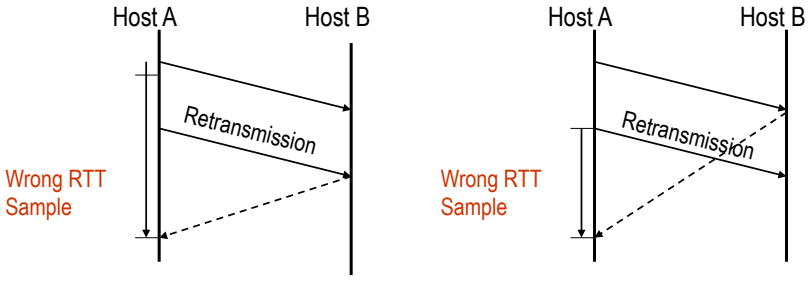
Early timeout
DUPLICATE
PACKETS

Sample RTT estimation



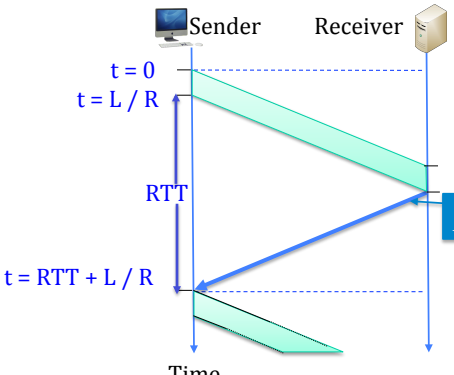
A flaw in this approach

- RTT estimation
  - Record time when TCP segment sent
  - Record time when ACK arrives and compute the difference
- Problem with retransmissions:



- Solution for retransmissions:
  - do not take samples, double the timeout value

Simple acknowledgement scheme



L = 1500 bytes * 8 = 12000 bits
R = 10 Gbps
RTT = 0.07 s (from NY to Beijing)
t = 0: first packet bit transmitted
t = L/R: last packet bit transmitted
t = RTT + L/R: ACK arrives

ACK

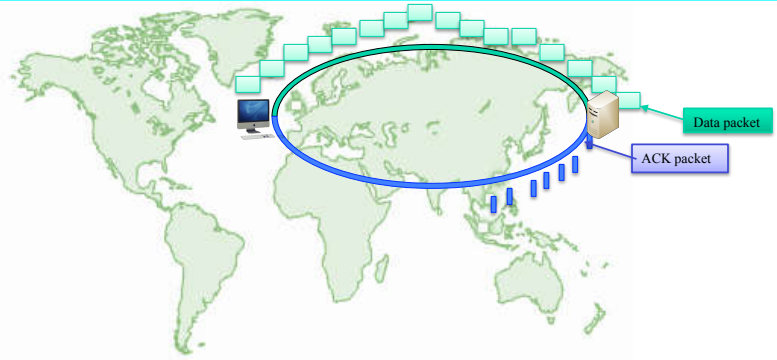
In the case of long RTT:
Transmission time for a message ≈ the number of packets of the message * RTT

$$\text{Link utilization} = \frac{L/R}{RTT + L/R} = \frac{(12000)*10^{-10}}{7.00012*10^{-2}} = 1.7*10^{-5}$$

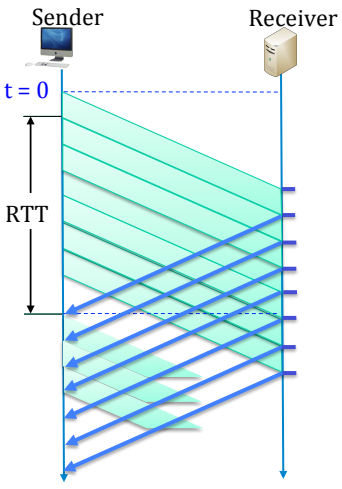
Link bandwidth is wasted by waiting!

Pipelined TCP

- Pipelining
  - Sender sends a series of packets without waiting for ACK
  - Sequence numbers are used
  - Flow control: do not exceed the buffer at receiver
  - Dramatically improves link utilization



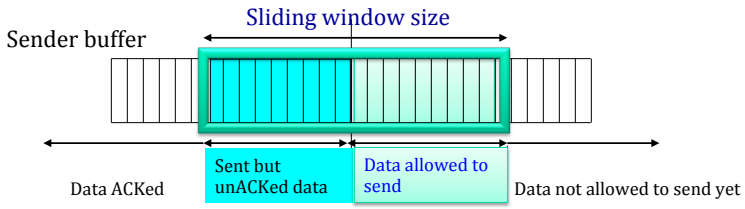
Pipelined transmission of packets



- Back-to-back packet transmission can improve the link utilization
- Queueing time of a message is reduced
- However, it is necessary to know how many packets to deliver for the first shot
  - Flow control: use the buffer size exchanged in 3-way handshake
- Use the ACK as the means of self-clocking to deliver the remaining packets

Pipeline and sliding window

- Flow Control by advertising available buffer size in the receiver
- Window (buffer) size
  - Amount that can be sent without acknowledgment
  - Receiver needs to be able to store this amount of data
- Receiver advertises the window size to sender
  - Tells the sender the amount of free space left
  - Sender shoots over the number of packets the receiver can digest



Outline

Part 4

- Overview
  - TCP, STCP and UDP
- TCP transport protocol
  - Reliable delivery
  - Flow control
  - TCP handshake
  - Congestion control

TCP header supports flow control

Flags: SYN
FIN
RST
PSH
URG
ACK

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

TCP flow control

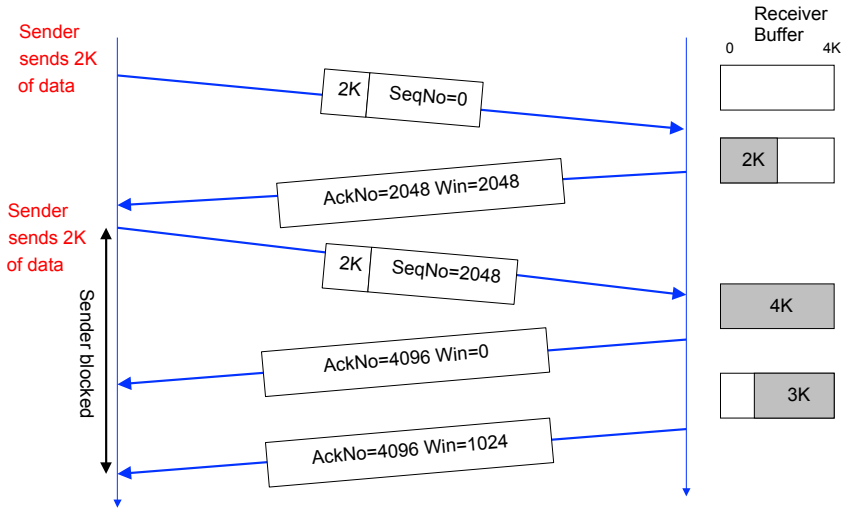
- The receiver returns two parameters to the sender

AckNo	window size (win)
32 bits	16 bits

- The interpretation is: I am ready to receive new data with

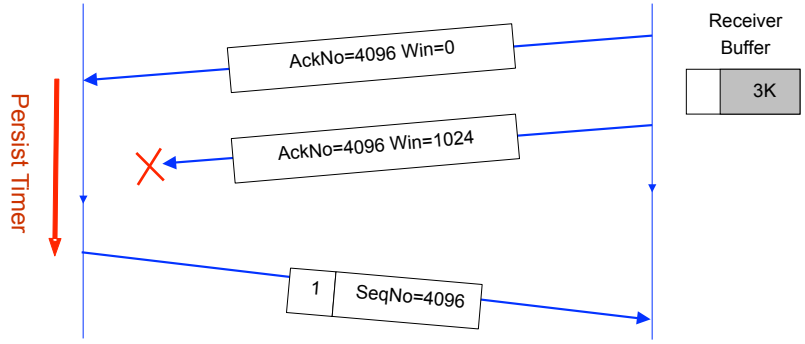
SeqNo= AckNo, AckNo+1, …., AckNo+win-1

Flow control example



TCP persist timer

- What if the last window update message is lost?
  - Receiver waiting for data, sender not allowed to send anything

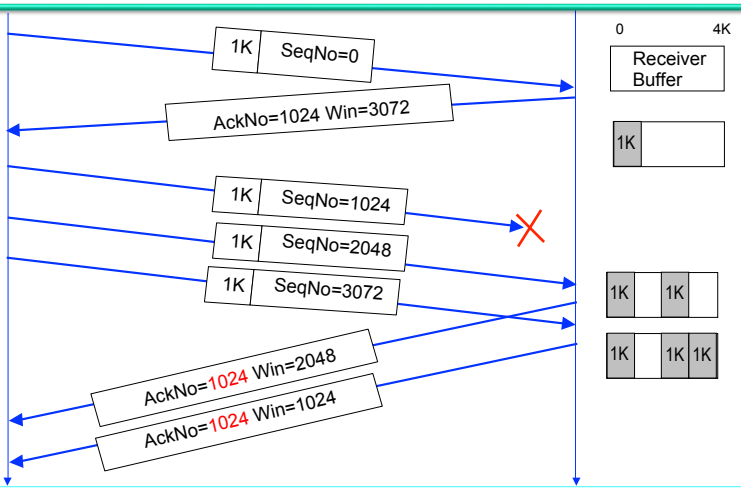


  - Sender sets persist timer when windows size goes to 0
  - When timer expires, sender sends a TCP segment with 1 byte of data (TCP probe)
  - If receiver still has window 0, it resend the ACK (does not cover the “illegal” byte)

Immediate Acknowledgment

- A TCP receiver should send an immediate ACK when the incoming segment fills in all or part of a gap in the sequence space
  - This will generate more timely information
  - TCP may generate an immediate acknowledgment (a duplicate ACK) when an out-of-order segment is received
  - One reason for doing so was the fast-retransmit algorithm
  - This duplicate ACK should not be delayed
- The purpose of this duplicate ACK is to let the other end know that a segment was received out of order, and to tell it what sequence number is expected

Immediate Acknowledgement Example



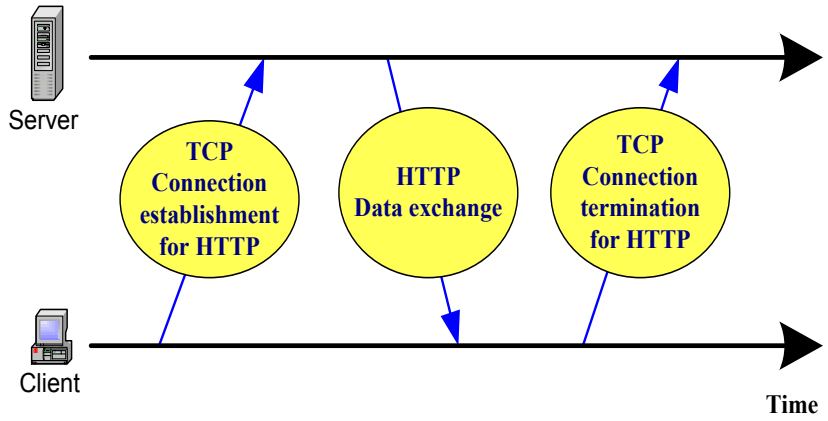
- Fast retransmission
  - Sender retransmits data after the triple duplicate ACK (before timeout)

Outline

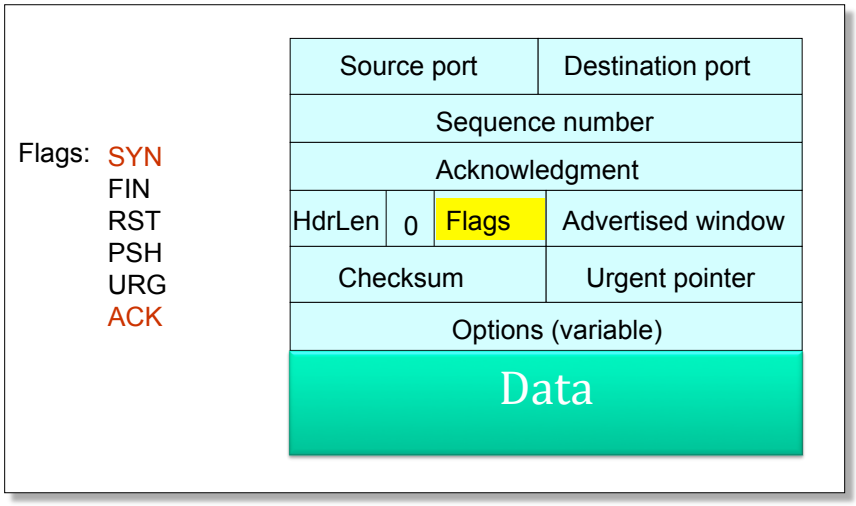
Part 4

- Overview
  - TCP, STCP and UDP
- TCP transport protocol
  - Reliable delivery
  - Flow control
  - TCP handshake
  - Congestion control

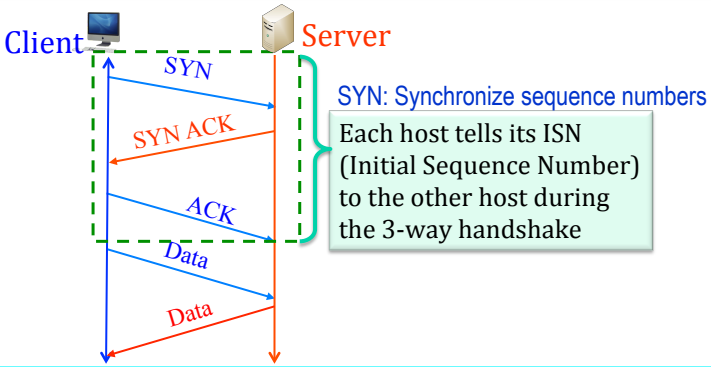
TCP connection setup and teardown



TCP Header

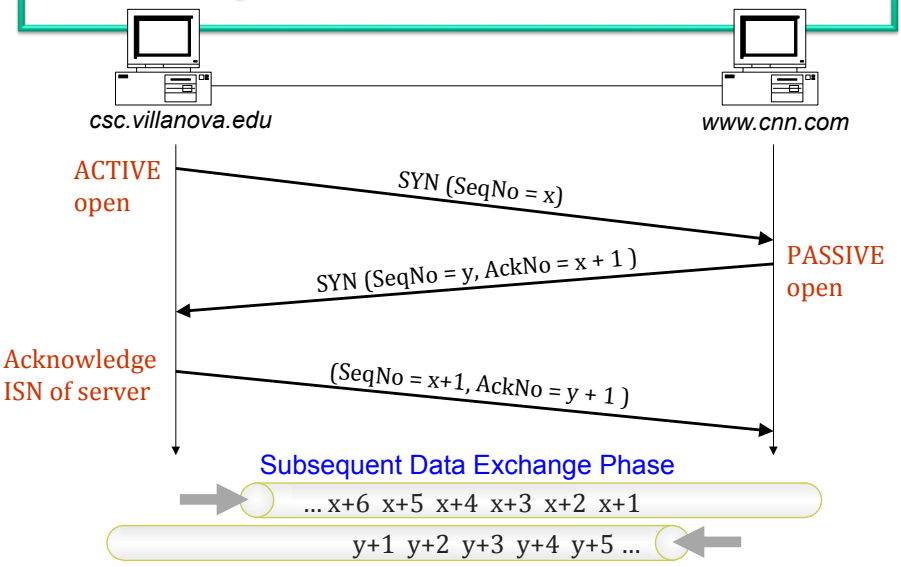


TCP handshake



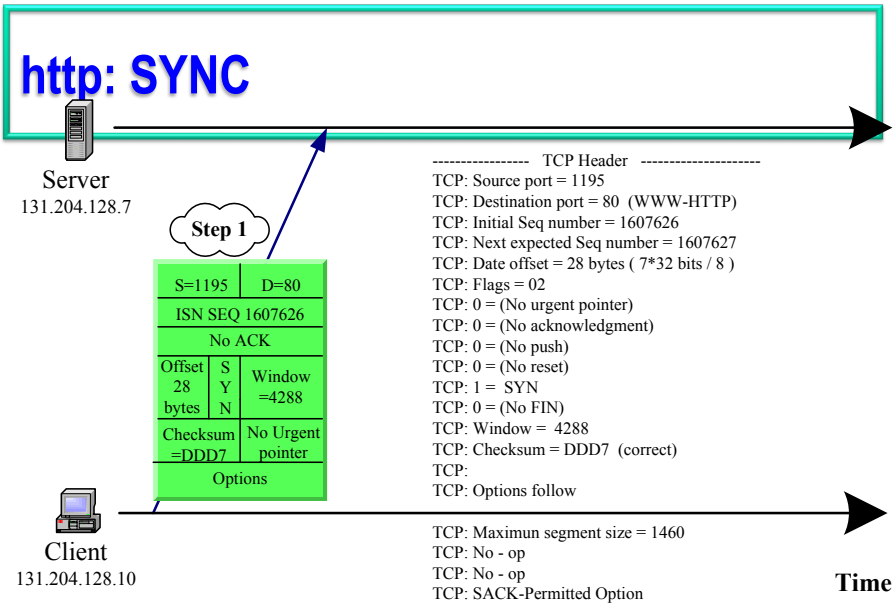
- Three-way handshake to establish connection
  - Client sends a SYN (open) to the server
  - Server returns a SYN acknowledgment (SYN ACK)
  - Client sends an ACK to acknowledge the SYN ACK

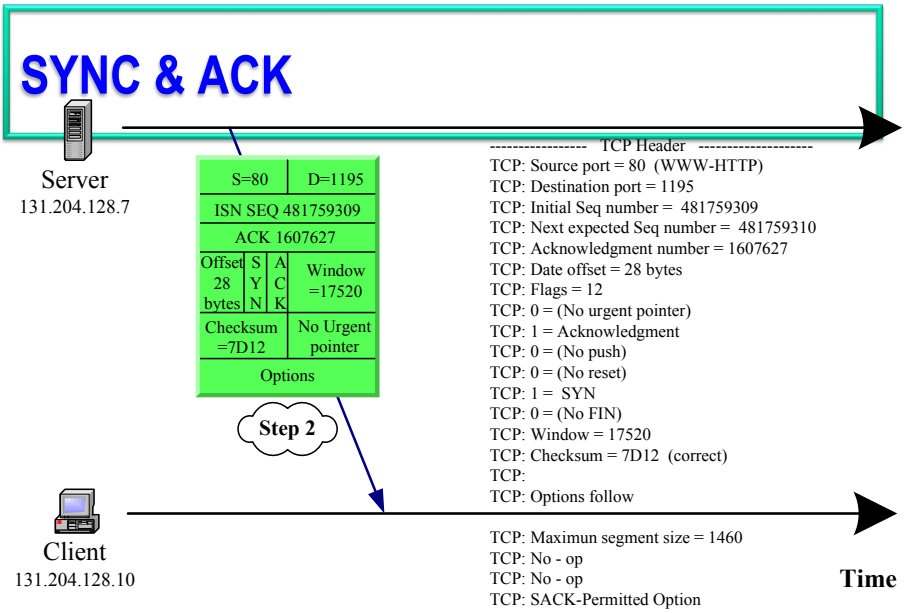
Three-Way Handshake



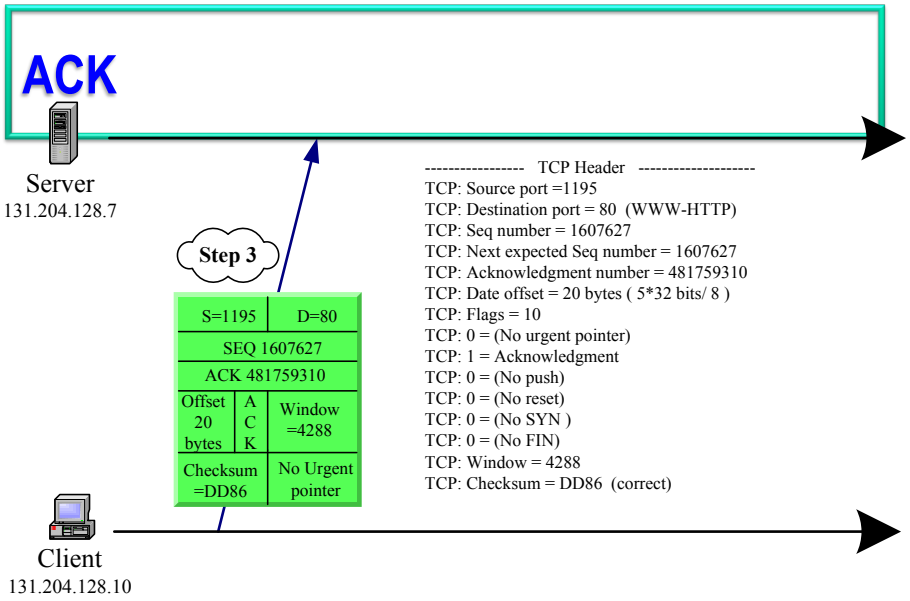
TCP Connection

- Three way handshake
  - Client host sends TCP SYN segment to server
    - Specifies client ISN
    - Advertises buffer size
    - No data
  - Server host receives SYN, replies with SYN ACK segment
    - Server allocates buffers (for flow control info)
    - Specifies server ISN
    - ACK # = client ISN + 1
  - Client receives SYN ACK, replies with ACK segment, which may contain data
    - RTT is estimated
    - ACK # = server ISN + 1





… upon receiving this packet, the client can start sending data

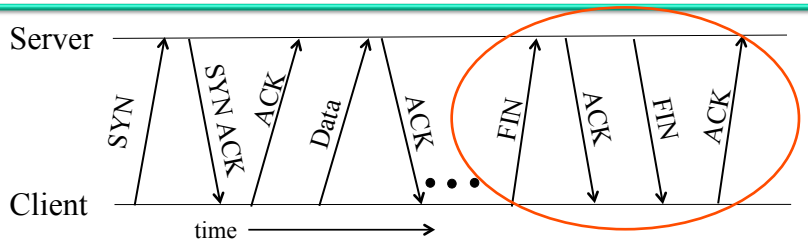


… upon receiving this packet, the server can start sending data

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 waits 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

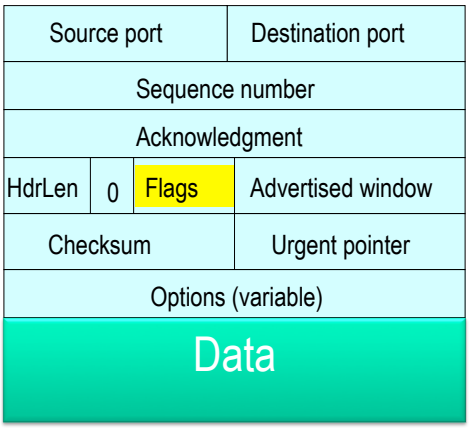
Tearing Down the Connection



- Closing (each end of) the connection
  - Client sends TCP FIN control segment to server
  - Server receives FIN, replies with ACK
  - Client receives FIN, replies with ACK
  - Server receives ACK. Connection closed

Other TCP Flags

SYN
FIN
RST – reset (abort) connection
PSH – send data out immediately
URG – marks some data as urgent
ACK



Outline

Part 4

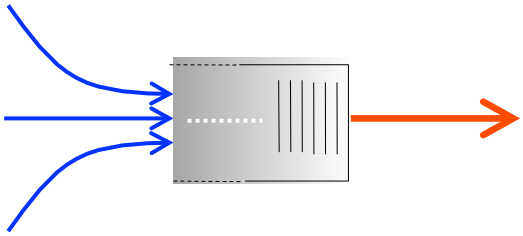
- Overview
  - TCP, STCP and UDP
- TCP transport protocol
  - Reliable delivery
  - Flow control
  - TCP handshake
  - Congestion control

IP best-effort design philosophy

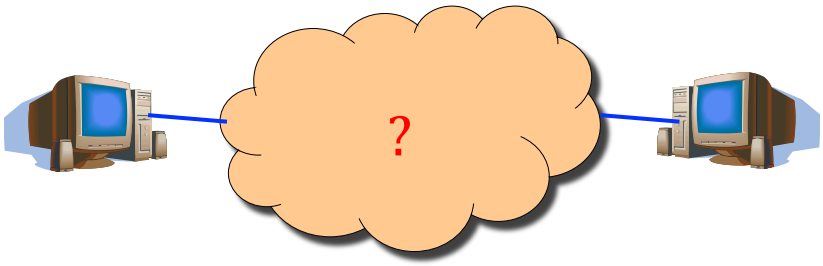
- Best-effort delivery
  - Let everybody send
  - Try to deliver what you can
  - … and just drop the rest

Congestion is Unavoidable

- Two packets arrive at the same time
  - The node can only transmit one
  - … and either buffer or drop the other
- If many packets arrive in short period of time
  - The node cannot keep up with the arriving traffic
  - … and the buffer may eventually overflow



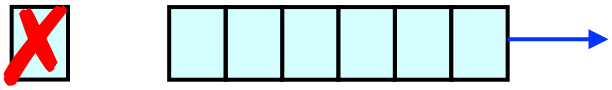
Detect and Respond to Congestion



- What do the end hosts see?
- What can the end hosts do to make good use of shared underlying resources?

How it looks to the end host

- Delay: packet experiences high delay
  - RTT estimate
- Loss: packet gets dropped along the way
  - Timeout and/or duplicate acknowledgements



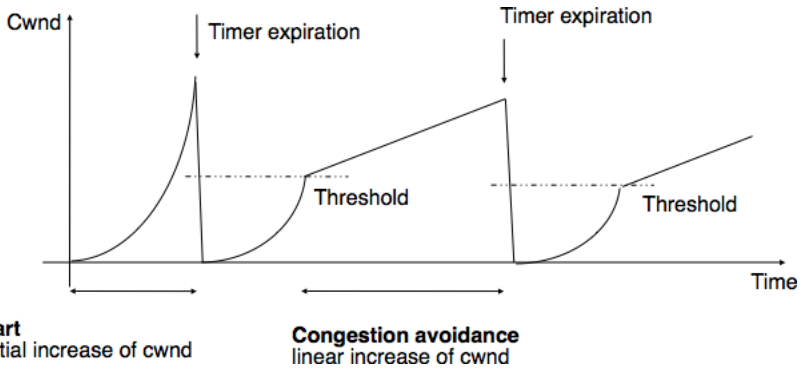
What can the end host do?

- Upon detecting congestion
  - Decrease the sending rate (e.g., divide in half)
  - End host does its part to alleviate the congestion
- But, what if conditions change?
  - Suppose there is more bandwidth available
  - Would be a shame to stay at a low sending rate
- Upon *not* detecting congestion
  - Increase the sending rate, a little at a time
  - And see if the packets are successfully delivered

TCP Congestion Window

- Each TCP sender maintains a congestion window
  - Maximum number of bytes to have in transit
  - i.e., number of bytes still awaiting acknowledgments
- Adapting the congestion window
  - Decrease upon losing a packet: backing off
  - Increase upon success: optimistically exploring
  - Always struggling to find the right transfer rate
- Both good and bad
  - Pro: avoids having explicit feedback from network
  - Con: under-shooting and over-shooting the rate

Leads to the TCP "Sawtooth"



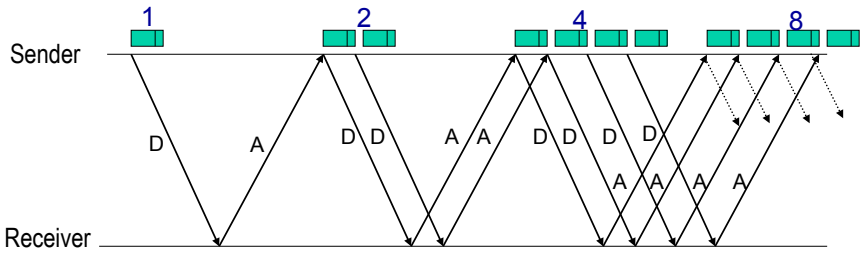
- If CWND is less than or equal to Threshold, TCP is in slow start
- Otherwise TCP is performing congestion avoidance

Slow Start

- Slow Start:
  - Start with congestion window equal to 1MSS
  - Increase congestion window by 1 MSS for each TCP segment successfully sent (each ACK received)
  - The rate increases exponentially
- Congestion Avoidance:
  - Increase congestion window by 1 MSS every RTT (for each FULL window of data successfully sent) until loss is detected

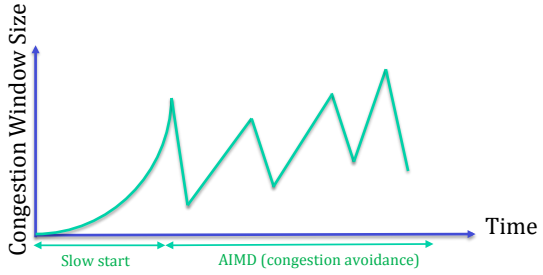
Slow start in action

Double CWND per round-trip time



Congestion avoidance

- Additive Increase:
  - Increase congestion window by 1 MSS every RTT (for each FULL window of data successfully sent) until loss is detected
- Multiplicative decrease
  - cut CWND in half after 3^{d} duplicate ACKs (congestion sign occurs)

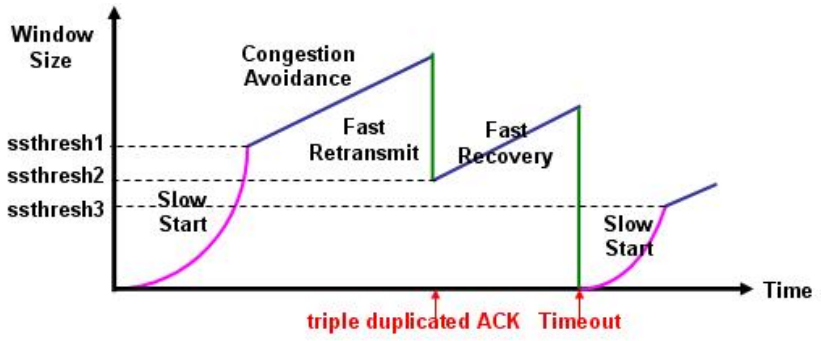


Sawtooth behavior: probing for available bandwidth

Two kinds of congestion signs in TCP

- Timeout
  - Packet n is lost and detected via a timeout
  - Reset treshold = CWND/2 and CWND = 1 MSS
- Triple duplicate ACK
  - Packet n is lost, but packets n+1, n+2, etc. arrive
  - Receiver sends duplicate acknowledgments
  - … and the sender retransmits packet n quickly
  - Reset CWND = CWND/2 and keep going

Triple duplicate ACK



- AIMD (Additive Increase, Multiplicative Decrease):
  - a necessary condition for stability of TCP

Fast retransmit and Fast recovery

- Fast retransmit
  - The fast retransmit algorithm uses the arrival of 3 duplicate ACKs (4 identical ACKs without the arrival of any other intervening packets) as an indication that a segment has been lost
  - After receiving 3 duplicate ACKs, TCP performs a retransmission
- After the fast retransmit algorithm, the Fast Recovery algorithm governs the transmission of new data until a non-duplicate ACK arrives
  - The reason for not performing slow start is that the receipt of the duplicate ACKs indicates that segments are most likely still reaching the receiver (since the receiver can only generate a duplicate ACK when a segment has arrived)

Receiver window vs. Congestion window

- Flow control
  - Keep a *fast sender* from overwhelming *a slow receiver*
- Congestion control
  - Keep a *set of senders* from overloading the *network*
- Different concepts, but similar mechanisms
  - TCP flow control: receiver window
  - TCP congestion control: congestion window
  - TCP window: min{congestion window, receiver window}

Question

- Suppose two hosts have a long-lived TCP session over a path, when a link on the path fails. What new congestion window size does the TCP sender use?

Exercise

- A TCP sender is using a MSS of 1000 bytes and the receiver's Advertised Window is 8000 bytes. Suppose that the TCP Congestion Window is equal to 4000 bytes just before a timeout occurs.

  After the timeout, six full segments of data – S1, S2, S3, S4, S5, S6 - are successfully sent and acknowledged. The seventh segment is lost and must be retransmitted.

  Show the values for Congestion Window (CW) and Threshold (TH) as they change with each arriving ACK and then after the retransmission of the seventh segment.

Event	Congestion Window	Congestion Threshold	Phase (slow start, cong. avoidance)
Timeout			
ACK for S1 arrives			
ACK for S2 arrives			
ACK for S3 arrives			
ACK for S4 arrives			
ACK for S5 arrives			
ACK for S6 arrives			
Timeout for S7			

Exercise

- Consider a TCP connection in which the TCP Advertised Window is equal to 6 Kbytes, the TCP Congestion Window is equal to 8Kbytes, the Maximum Segment Size (MSS) is 1Kbyte and all transmitted packets are 1Kbyte in length.
  1. How many packets could be in traffic unacknowledged yet?
  2. If a timeout occurs, how many packets it takes to send and get acknowledged to get the Congestion Window up to 8Kbytes again? Assume no transmission errors.