Transport Layer Protocols

ELEC-C7420 Basic Principles in Networking

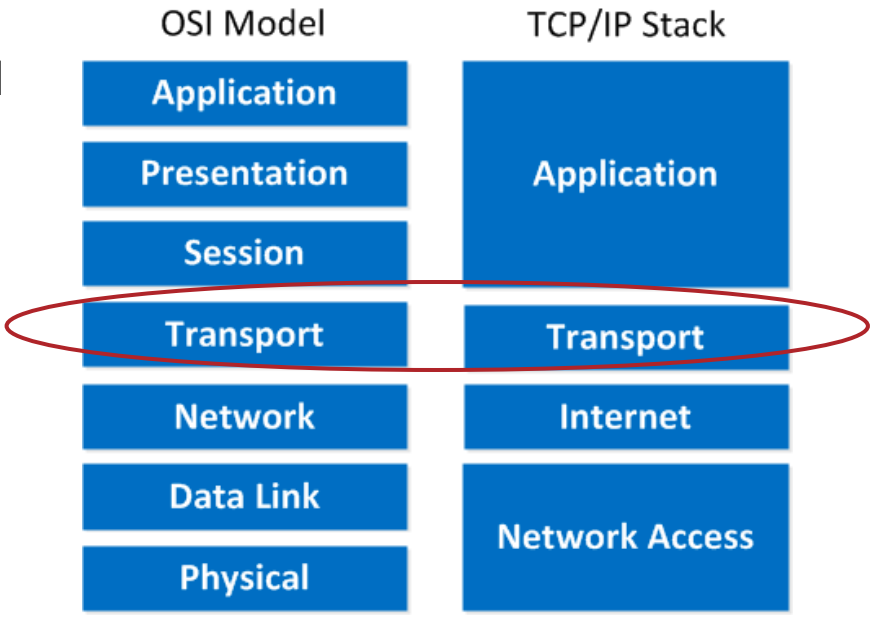
Aalto-yliopisto
Sähkötekniikan
korkeakoulu

Yu Xiao

28.1.2019

Transport Layer

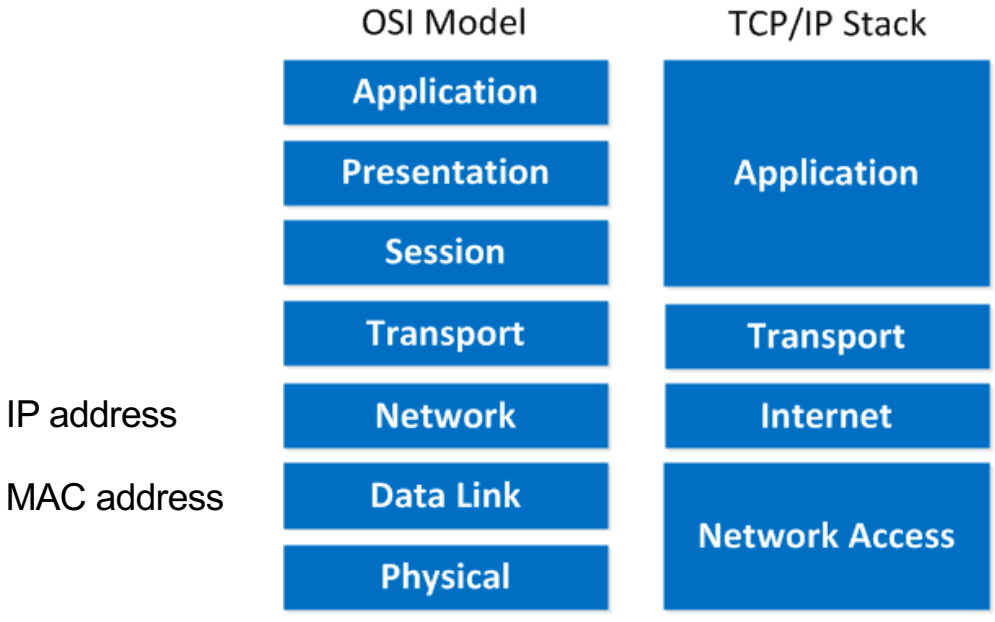
- **Define how data is delivered from source to destination**
- **Two major protocols:**
  - TCP: Transmission Control Protocol
  - UDP: User Datagram Protocol



Learning Outcomes

After this lecture, you should be able to

- **Describe the working mechanisms of two transport layer protocols including TCP and UDP**
- **Understand the differences between TCP and UDP**



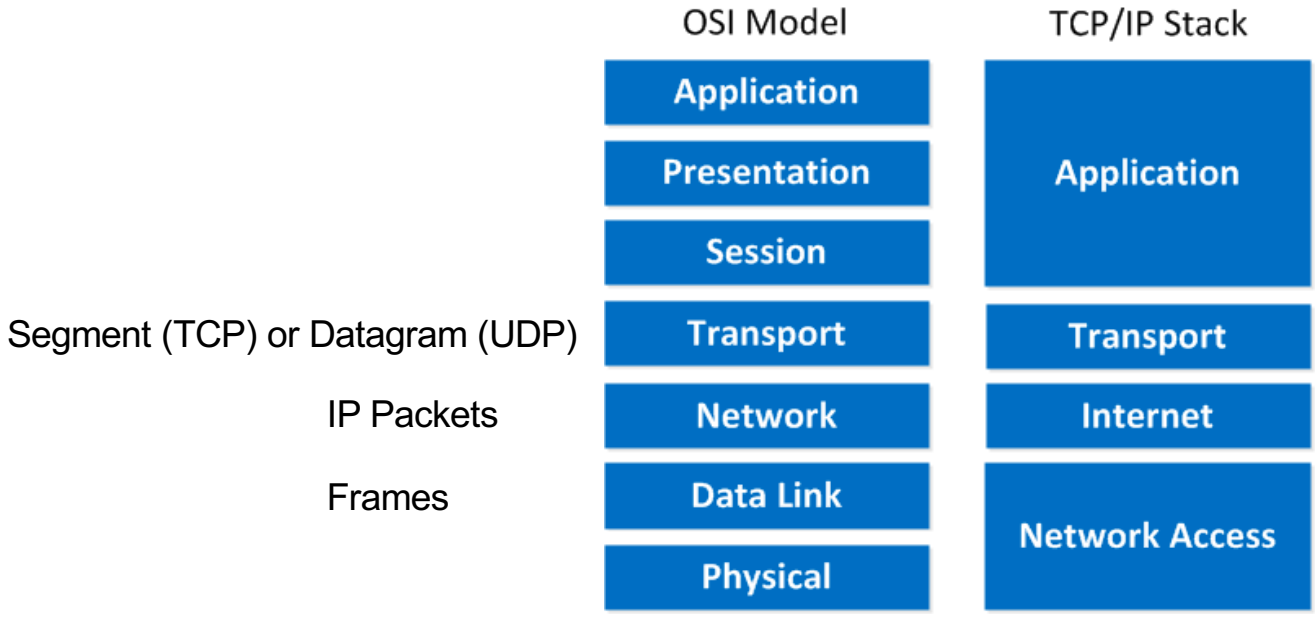
IP packets are addressed to a host. How to decide which application gets which packets?

TCP/UDP Ports

- TCP/UDP extends host-to-host delivery to process-to-process delivery
- Port <-> server process
- Ports are identified for each protocol and address combination by 16-bit unsigned numbers, known as port number.
- One IP address → 65535 TCP Ports and another 65535 UDP Ports
- Socket: <host, port>
- Socket address: <IP address, port number>, e.g. 192.168.0.10:80
- Client must know server's port

Port Number Ranges and Well Known Ports

- **Port numbers 0 – 1023 are well-known ports.** These are allocated to server services by the Internet Assigned Numbers Authority (IANA), e.g. web servers normally use port 80 and SMTP servers use port 25.
- **Ports 1024-49151** – Registered Port: these can be registered for services with the IANA and should be treated as semi-reserved.
- **Ports 49152-65535**: these are used by client programs and you are free to use these in client programs.


OSI Model
TCP/IP Stack
Application
Presentation
Session
Transport
Network
Data Link
Physical
Application
Transport
Internet
Network Access
Segment (TCP) or Datagram (UDP)
IP Packets
Frames

IP

IP provides unreliable service. It makes its best effort to deliver segments between communicating hosts, but it makes no guarantees.

- It does not guarantee segment delivery
- It does not guarantee orderly delivery of segments
- It does not guarantee the integrity of the data in the segments

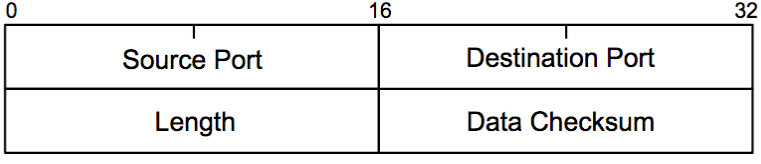
TCP	UDP
Connection-oriented	Connectionless
Reliable (Guaranteed delivery)	Best-effort delivery

UDP

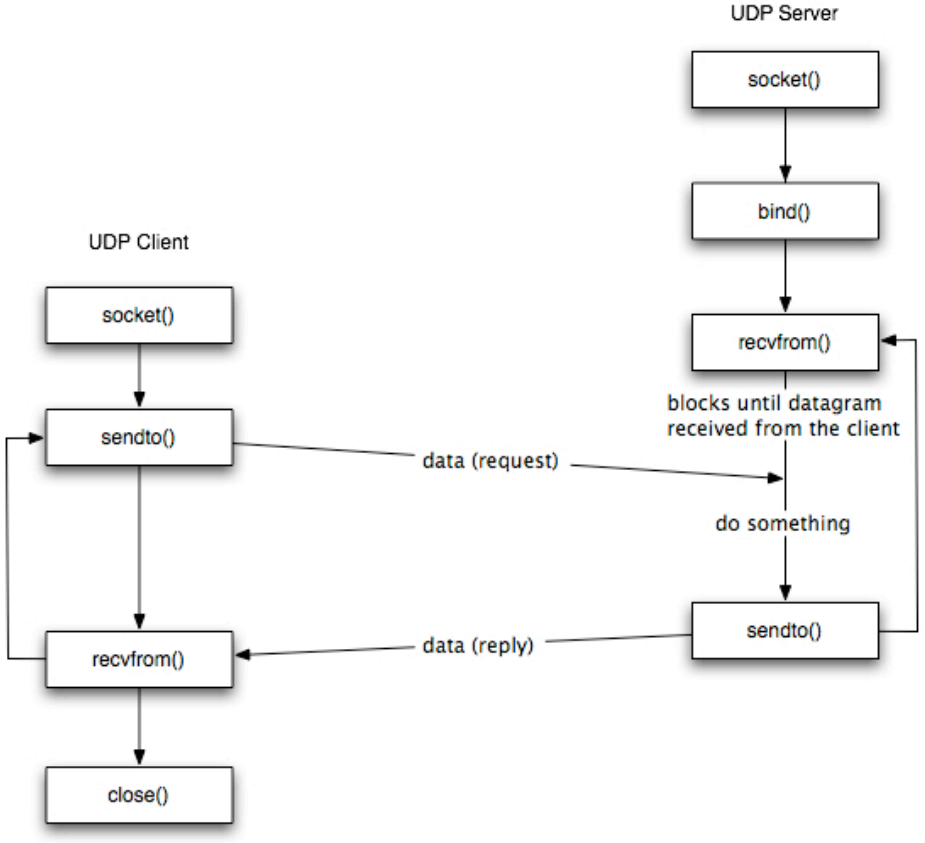
- UDP is **unreliable**, in that there is no UDP-layer attempt at timeouts, acknowledgment and retransmission; applications written for UDP must implement these.
- UDP is also **unconnected**, or stateless; if an application has opened a port on a host, any other host on the Internet may deliver packets to that ⟨host,port⟩ socket without preliminary negotiation.
- UDP packets can be dropped due to queue overflows either at an intervening router or at the receiving host.

UDP

- **Support multicast and broadcast**
- **Low overhead**
- **Commonly used for real-time apps**



Socket(): socket creation
Bind(): binds the socket to specified address and port number



Aalto-yliopisto
Sähkötekniikan korkeakoulu

Source:
https://www.cs.dartmouth.edu/~campbell/cs60/socketprogramming.html

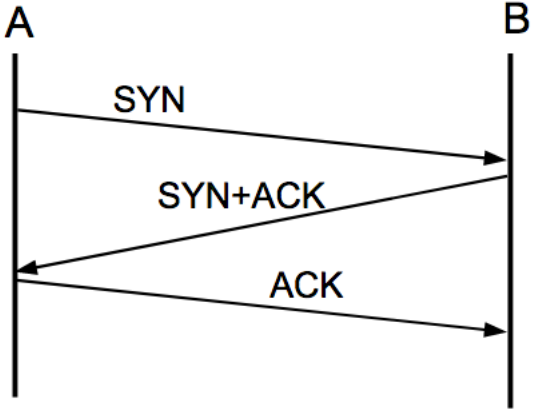
TCP extends IP with the following features

- Connection-orientation
- Stream-orientation
- Reliability
- Throughput management

Connection-oriented

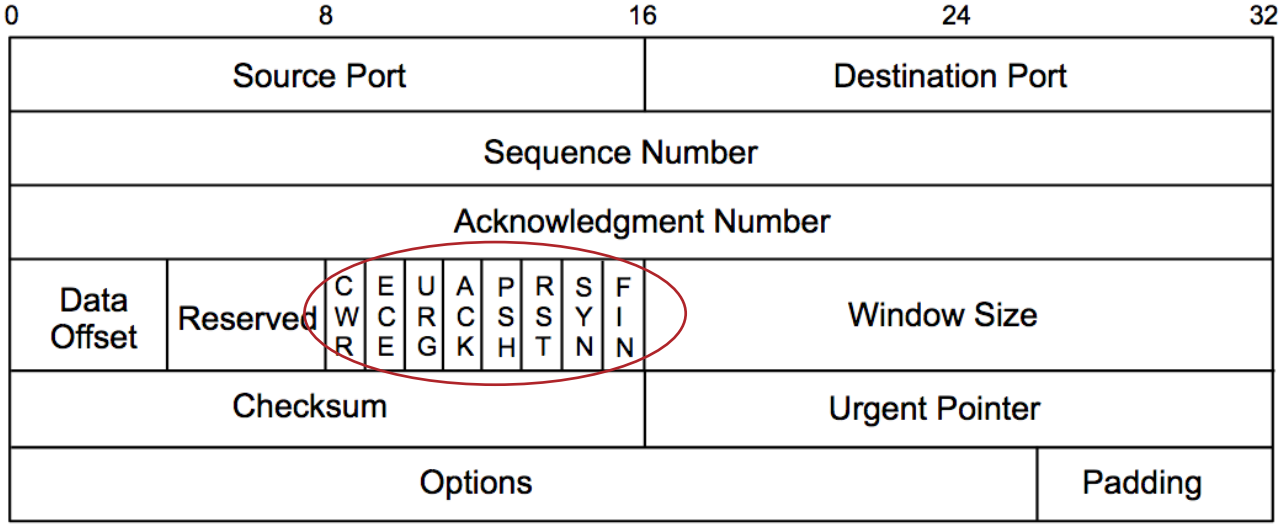
- Before one application process can begin to send data to another, the two processes must “handshake” with each other
- The process that is initiating the connection is called the *client process*, while the other process is called the *server process*
- **Three-way Handshake**

  1) A sends B a packet with the SYN bit set
  2) B responds with a SYN packet of its own; the ACK bit is also set.
  3) A responds to B’s SYN with its own ACK.



TCP three-way handshake

TCP Header


0
8
16
24
32
Source Port
Destination Port
Sequence Number
Acknowledgment Number
Data Offset
Reserved
CWR
ECE
URG
ACK
PSH
RST
SYN
FIN
Window Size
Checksum
Urgent Pointer
Options
Padding

Flag Bits

SYN: for synchronization; marks packets that are part of the new-connection handshake

ACK: indicates that the value carried in the acknowledge field is valid; that is, the segment contains an acknowledgement for a segment that has been successfully received.

FIN: For Finish; marks packets involved in the connection closing

RST: for Reset; indicates various error conditions

PSH: for Push; marks non-full packets that should be delivered

URG: for Urgent;

CWR and ECE: part of the explicit congestion notification mechanism

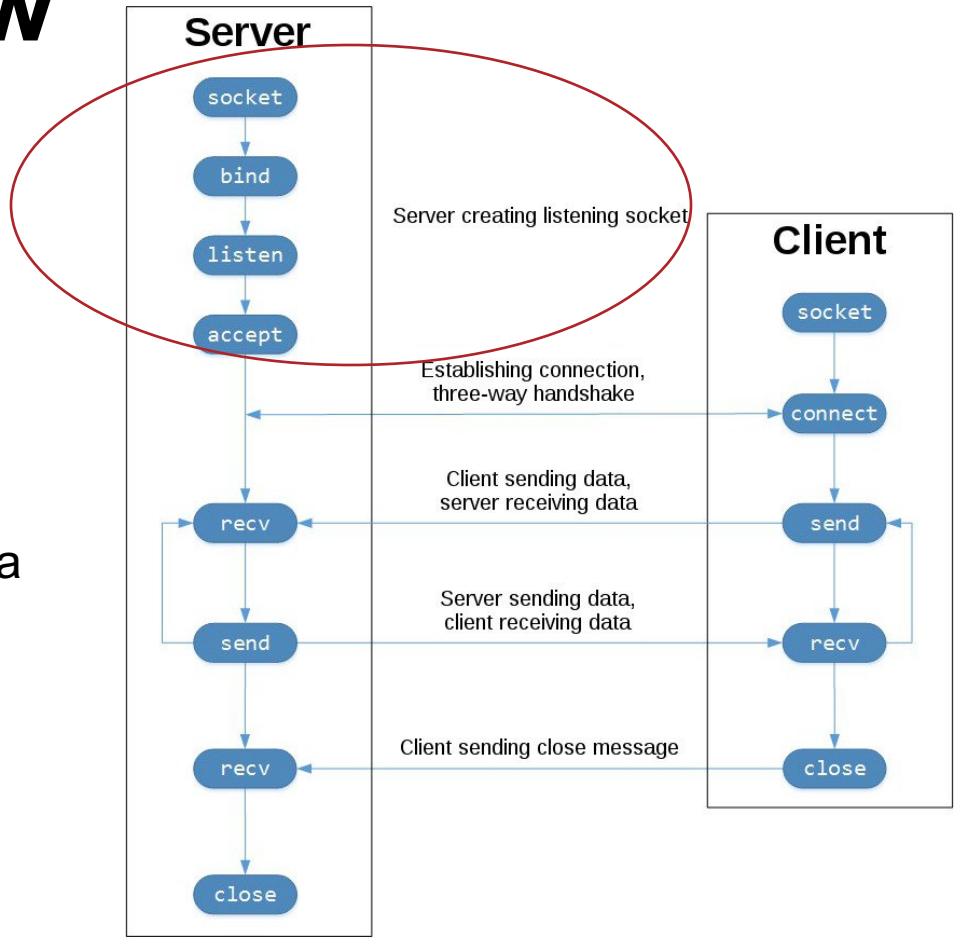
TCP Socket Flow

Socket(): socket creation

Bind(): binds the socket to specified address and port number

Listen(): puts the server socket in a passive mode, where it waits for the client to approach the server to make a connection

Accept(): extracts the first connection request on the queue of pending connections for the listening socket, creates a new connected socket



TCP Connection:

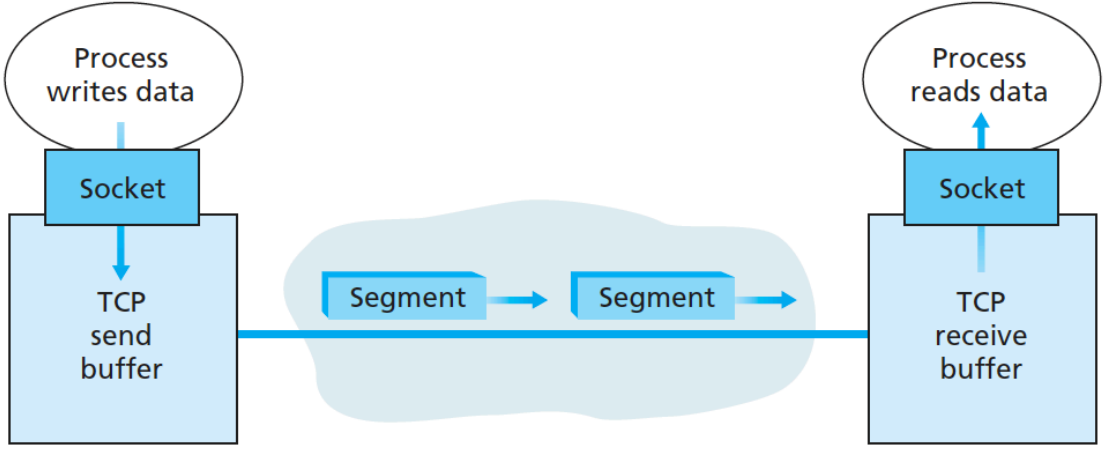
<Client IP, Client TCP Port, Server IP, Server TCP Port>

- **Client port numbers are dynamically assigned, and can be reused once the session is closed.**
- **There can be multiple TCP connections between two hosts. These connections should use different TCP Ports.**

Data Exchange via TCP

- **A TCP connection provides a full-duplex service**
- **A TCP connection is always point-to-point between a single sender and a single receiver**

How to determine Maximum Segment Size (MSS)?
Do you remember what Maximum Transmission Unit (MTU) is?

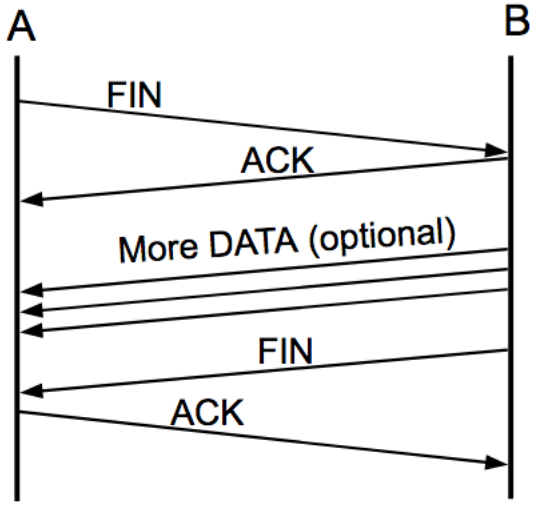


Source: Computer Networking – A top-down approach. Chapter 3.5

How to close a TCP connection?

Two Two-way FIN/ACK Handshake

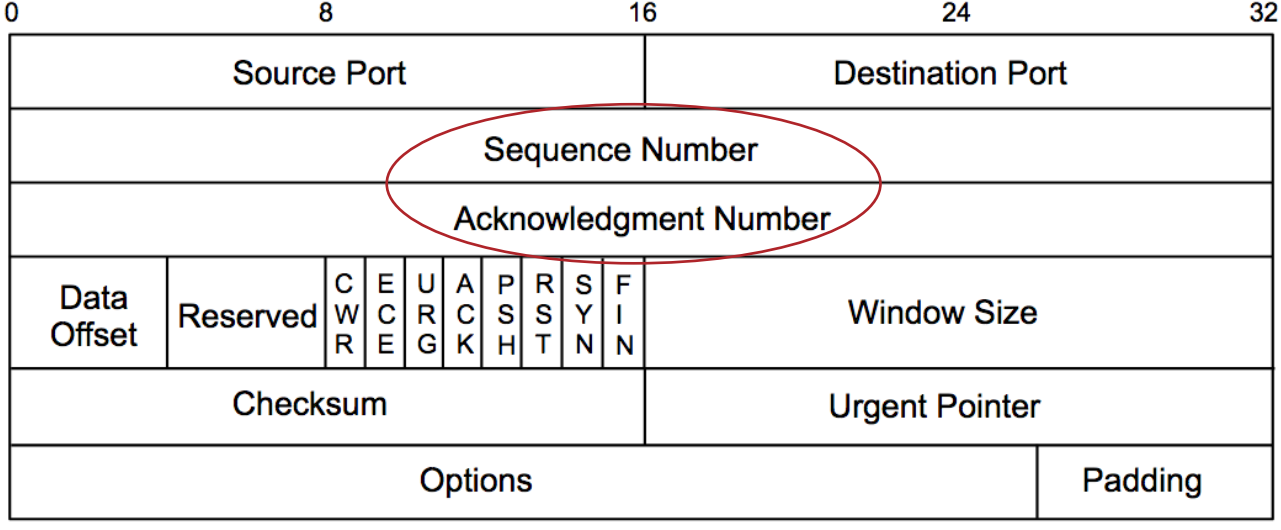
- A sends B a packet with the FIN bit set
- B sends A an ACK of the FIN
- B may continue to send additional data to A
- When B is also ready to cease sending, it sends its own FIN to A
- A sends B an ACK of the FIN; this is the final packet in the exchange



A typical TCP close

Reliability

- **TCP numbers each packet, and keeps track of which are lost and retransmits them after a timeout.**

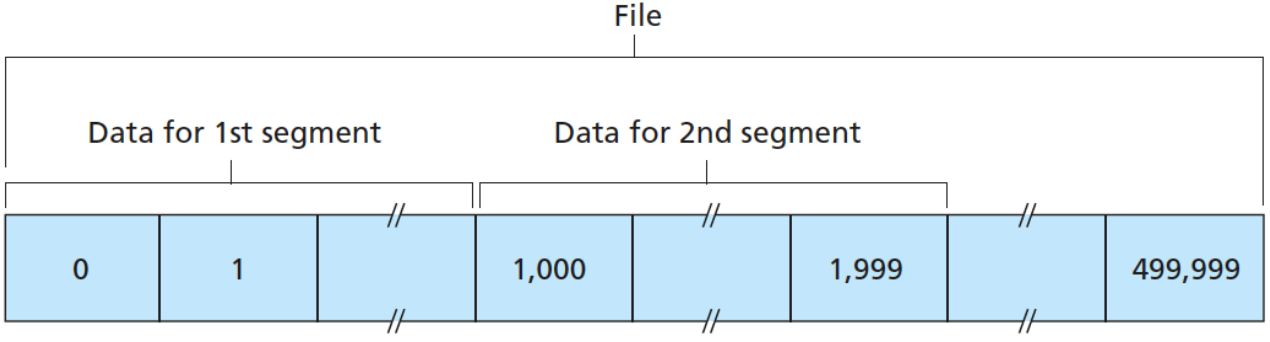


Sequence and Acknowledgement Numbers

- Numbering the data at the byte level
- Initial Sequence Number (ISN) is fixed for the lifetime of the connection. Each direction of a connection has its own ISN.
- The value of the Sequence Number, in relative terms, is the *position of the first byte of the packet in the data stream*, or the position of what would be the first byte in the case that no data was sent
- The value of the Acknowledgement Number, in relative terms, *represents the byte position for the next byte expected'*
- The sequence and acknowledgment numbers, as sent, represent these relative values *plus ISN*

Sequence Number

Example: MSS = 1000 Bytes, File size = 500,000 Bytes



If ISN = 0, sequence number of each segment = 0, 1000, 2000, …

Sequence Number

- **Both sides of a TCP connection randomly choose an ISN**. It may be any value between 0 and 4,294,967,295, inclusive.

- Protocol analyzers like Wireshark will typically display *relative* sequence and acknowledgement numbers in place of the actual values.

Acknowledgement Numbers

Examples:

1) Suppose that Host A has received all bytes numbered 0 through 535 from B and suppose that it is about to send a segment to Host B. *What should be the acknowledge number of the segment it sends to B?*
2) Suppose that Host A has received one segment from Host B containing bytes 0 through 535 and another segment containing bytes 900 through 1,000. For some reason Host A has not received bytes 536 through 899. In the next segment sent from A to B, *what should be the value of the acknowledge number field?*

- **TCP provides cumulative acknowledgements (i.e. TCP only acknowledges bytes up to the first missing byte in the stream)**

Out-of-Order Segment

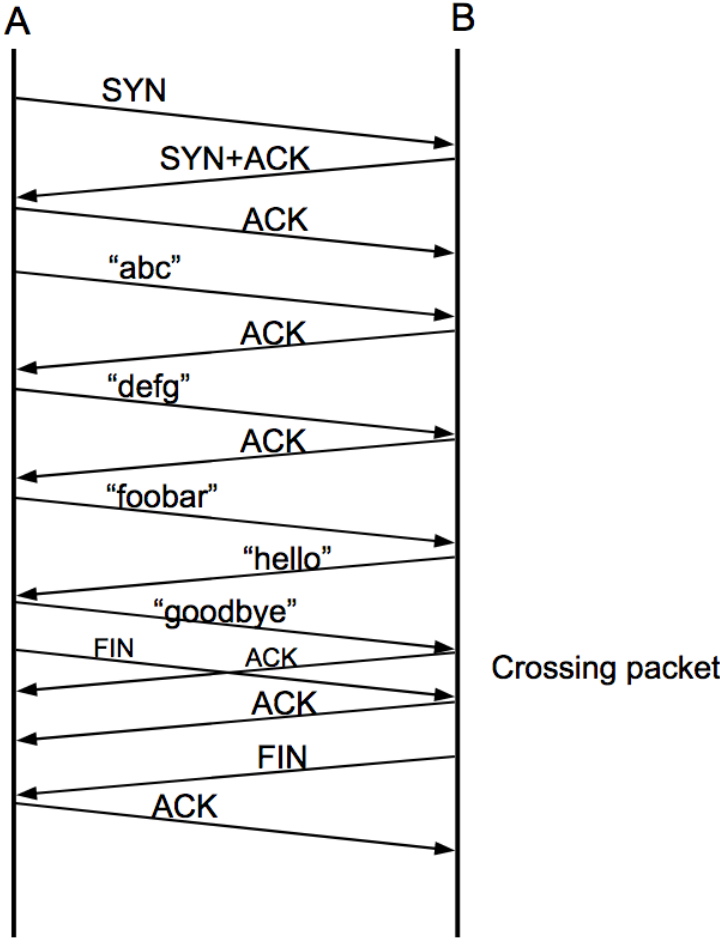
- Host A received the third segment (bytes 900 through 1,000) before receiving the second segment (bytes 536 through 899). Thus, the third segment arrived out of order. The subtle issue is: *What does a host do when it receives out-of-order segments in a TCP connection?*

Exercise

In terms of the sequence and acknowledgment numbers, SYNs count as 1 byte, as do FINs.

Assume that ISN = 0 on both sides

Can you calculate the seq and ack values of each segment?



Answers

	A sends	B sends
1	SYN, **seq=0**	
2		SYN+ACK, seq=0, **ack=1** (expecting)
3	ACK, **seq=1**, ack=1 (ACK of SYN)	
4	“abc”, **seq=1**, ack=1	
5		ACK, seq=1, **ack=4**
6	“defg”, **seq=4**, ack=1	
7		seq=1, **ack=8**
8	“foobar”, **seq=8**, ack=1	
9		seq=1, **ack=14**, “hello”
10	**seq=14**, ack=6, “goodbye”	
11,12	**seq=21**, ack=6, FIN	seq=6, **ack=21** ;; ACK of “goodbye”, crossing packets
13		seq=6, **ack=22** ;; ACK of FIN
14		seq=6, **ack=22**, FIN
15	**seq=22**, ack=7 ;; ACK of FIN	

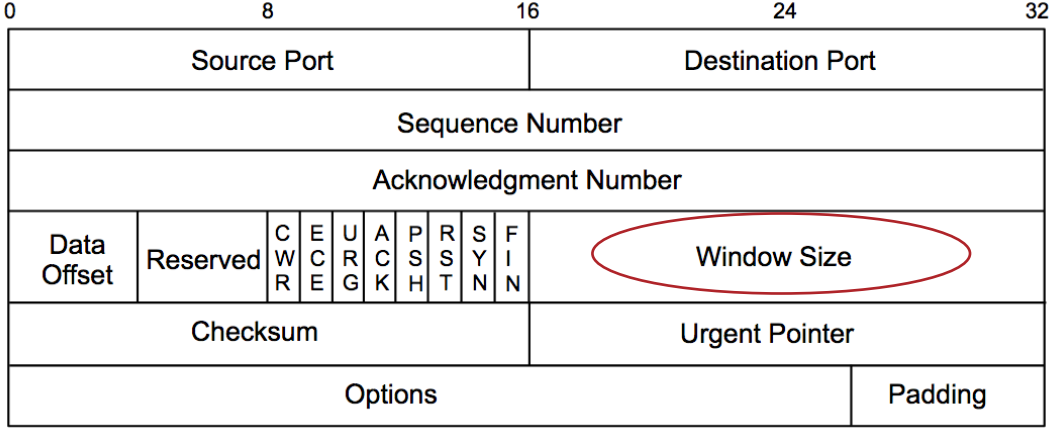
Source: http://intronetworks.cs.luc.edu/current/html/tcp.html#tcp-header
Chapter 12.3

TCP Timeout and Retransmission

- When TCP sends a segment containing user data (this excludes ACK-only packets), it sets a timer. If that timer expires before the segment data is acknowledged, the segment is retransmitted.
- The length of timeout interval is adapted to RTT
- Acknowledgements are sent for every arriving data packet (unless Delayed ACKs are implemented)

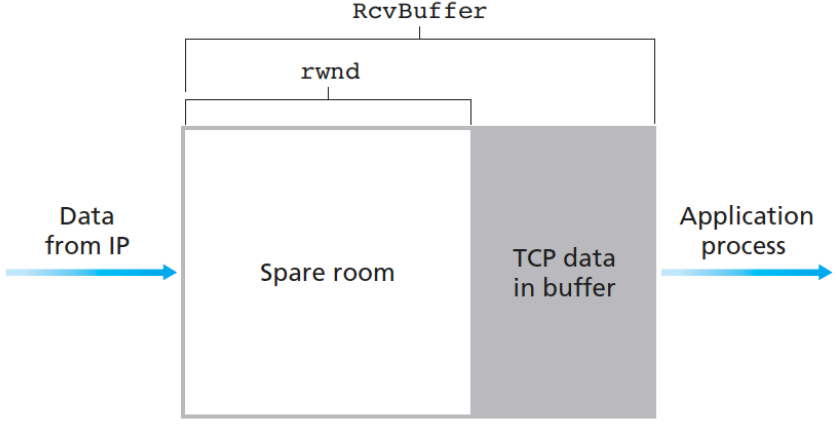
TCP Flow Control

- **Window size indicates the number of bytes that a receiver is willing to accept.**
- Flow-control service is used for eliminating the possibility of the sender overflowing the receiver's buffer



Example Scenario

- Suppose that Host A is sending a large file to Host B over a TCP connection. Host B allocates a receive buffer to this connection; denote its size by ***RcvBuffer*** .
- ***LastByteRecv***: the number of the last byte in the data stream that has arrived from the network and has been placed in the receive buffer at B
- ***LastByteRead***: the number of the last byte in the data stream read from the buffer by the app process in B



Receive Window

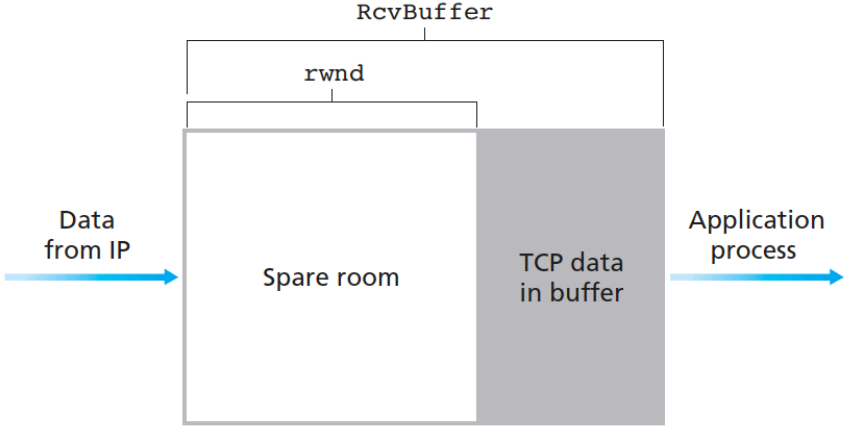
- **Sender maintains a variable called the receive window**
- TCP is not permitted to overflow the allocated buffer

$$LastByteRecv - LastByteRead \leq RcvBuffer$$

- The receive window rwnd is set to the amount of spare room in the buffer:

$$rwnd = RcvBuffer - [LastByteRecv - LastByteRead]$$

Because TCP is full-duplex, the sender at each side of the connection maintains a distinct receive window.

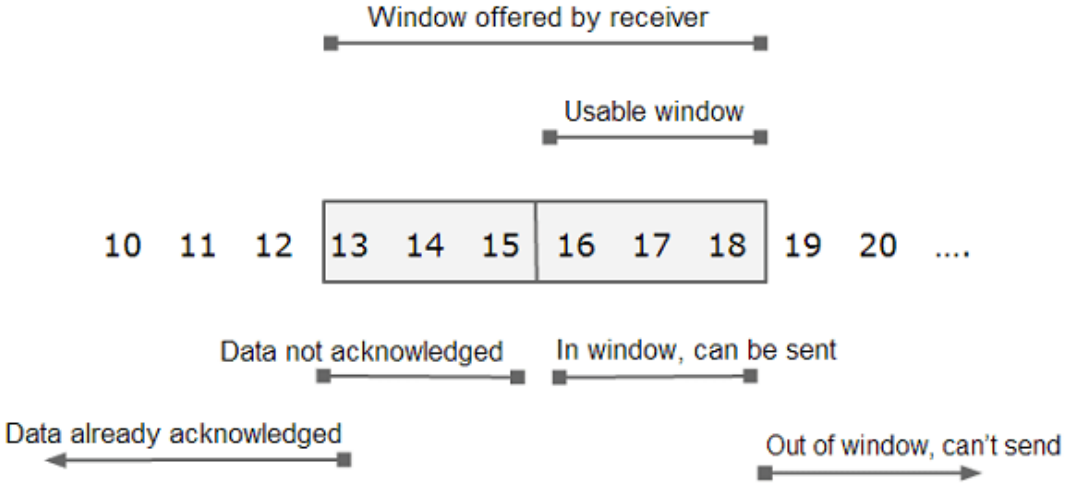


How does the connection use the variable *rwnd* to provide the flow-control service?

Sliding Window

- Host B tells Host A how much spare room it has in the connection buffer by placing its current value of *rwnd* in the receive window field of every segment it sends to A. Initially, Host B sets rwnd = RcvBuffer.

This window slides towards right depending upon how fast receiver consumes data and sends acknowledgement and hence known as sliding window.



Source: https://www.thegeekstuff.com/2013/10/tcp-sliding-window/

TCP Window Scale Option

- **TCP window scale option is needed for efficient transfer of data when the bandwidth-delay product is greater than 64KB.**

  E.g. 1.5Mbps x 0.513s = 96 KB

- **Effective window size = receive window size x window scale**

Why do we need congestion control?

Too many sources attempting to send data at too high a rate

→ Longer delay, packet loss

TCP Congestion Control

- **End-to-end congestion control,** since IP layer provides no explicit feedback to end systems regarding congestion
- **Each sender limits the rate at which it sends traffic into its connection as a function of perceived network congestion**
  - ✓ If a TCP sender perceives that there is little congestion on the path between itself and the destination, then the TCP sender increases its send rate;
  - ✓ If the sender perceives that there is congestion along the path, then the sender reduces its send rate.

1) How does a TCP sender limit the rate at which it sends traffic into its connection?
2) How does a TCP sender perceive that there is congestion on the path between itself and the destination?
3) What algorithm should the sender use to change its send rate as a function of perceived end-to-end congestion?

Congestion Window

- **Sender maintains a variable called congestion window, denoted *cwnd.***
- **The amount of unacknowledged data at a sender may not exceed the minimum of cwnd and rwnd**

$$\text{LastByteSent} - \text{LastByteAcked} \leq \min\{\text{cwnd}, \text{rwnd}\}$$

- **A lost segment implies congestion, and hence, the TCP sender's rate should be decreased when a segment is lost.**
  - **Loss events**: a timeout event, receipt of duplicated acks for a given segment
- **TCP uses acknowledges to trigger (or clock) its increase in congestion window size**

What would happen if senders collectively send too fast or too slowly?

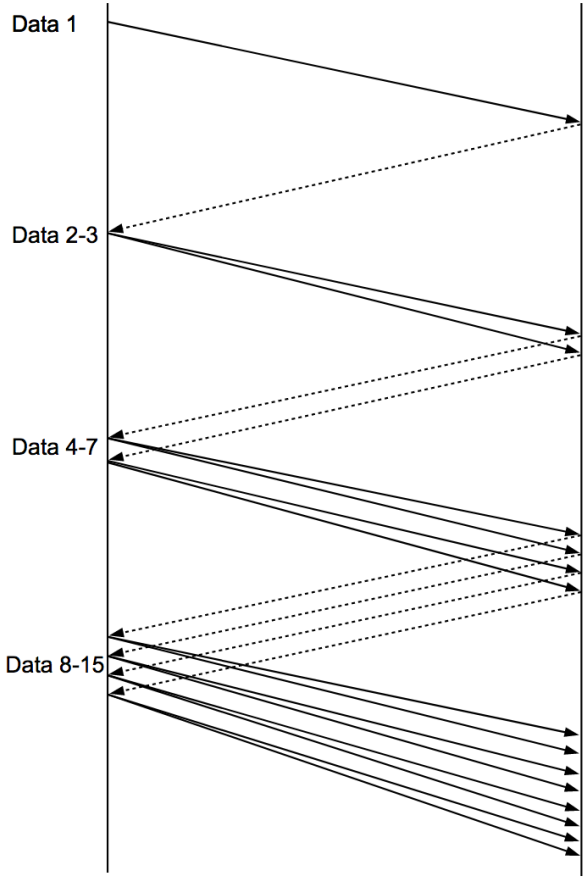
TCP Congestion-Control Algorithm

Three Major Components:

- **Slow Start**
- **Congestion Avoidance**
- **Fast recovery (recommended for TCP senders, but not required)**

Slow Start

- Set initial cwnd = 1
- cwnd = cwnd x 2 after receiving cwnd ACKs
- The TCP send rate starts slow but grows exponentially during the slow start phase



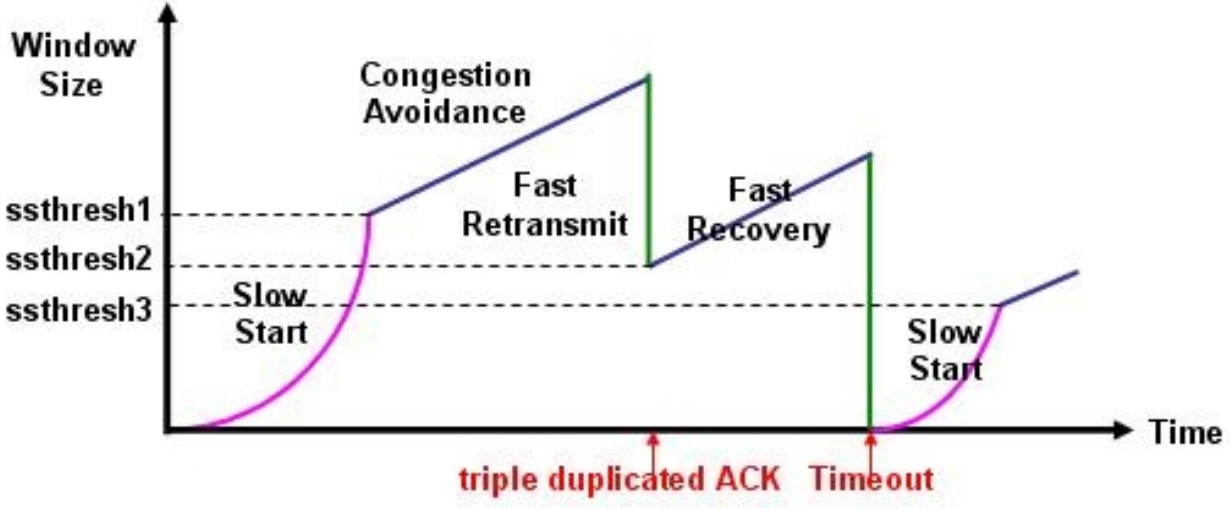
Slow Start with discrete packet flights

When should the exponential growth end?

- **If there is a loss event indicated by a timeout, the TCP sender sets the value of *cwnd* to 1 and begins the slow start anew. Also set *ssthresh* (‘slow start threshold’) to *cwnd/2***

When should the exponential growth end?

When *cwnd* reaches or surpasses *ssthresh*, slow start ends and TCP transitions into congestion avoidance mode



Source: https://sites.google.com/site/projectcodebank/computer-engineering-notes/tcp-congestion-control-algorithm

Congestion Avoidance

- cwnd = cwnd + 1 MSS whenever a new ACK arrives
- If congestion was indicated by a timeout cwnd is reset to 1 segment, which automatically puts the sender into slow start mode
- If congestion was indicated by duplicate Acknowledgements the fast re-transmit and fast recovery algorithms are invoked.

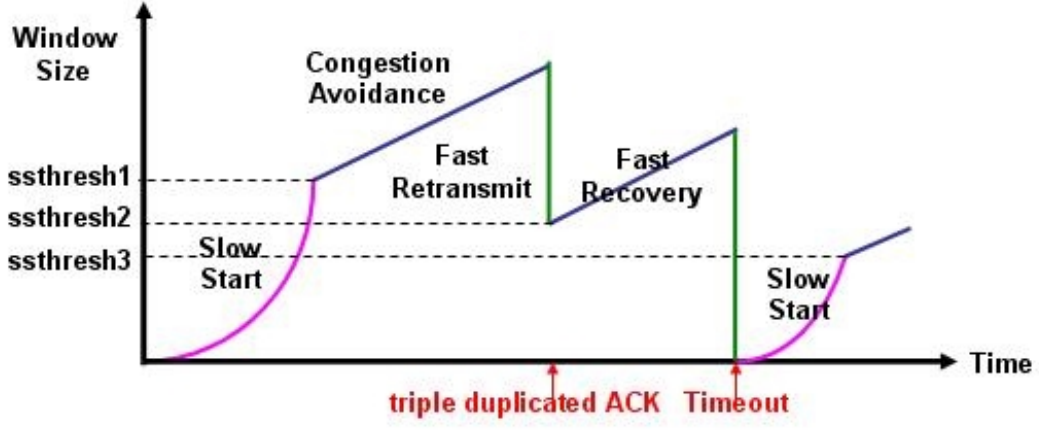


Figure 24.32: Example of Taho TCP

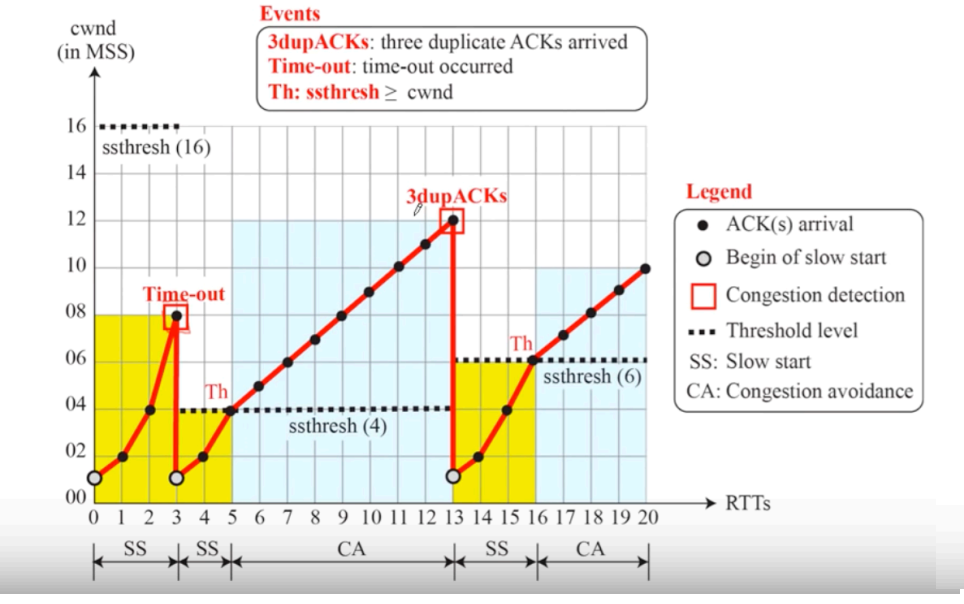
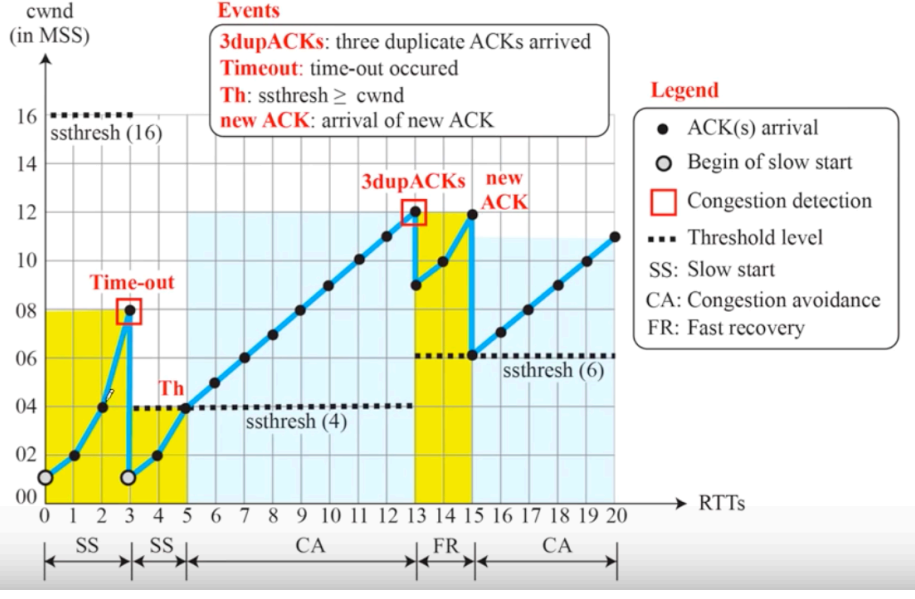


Figure 24.34: Example of a Reno TCP

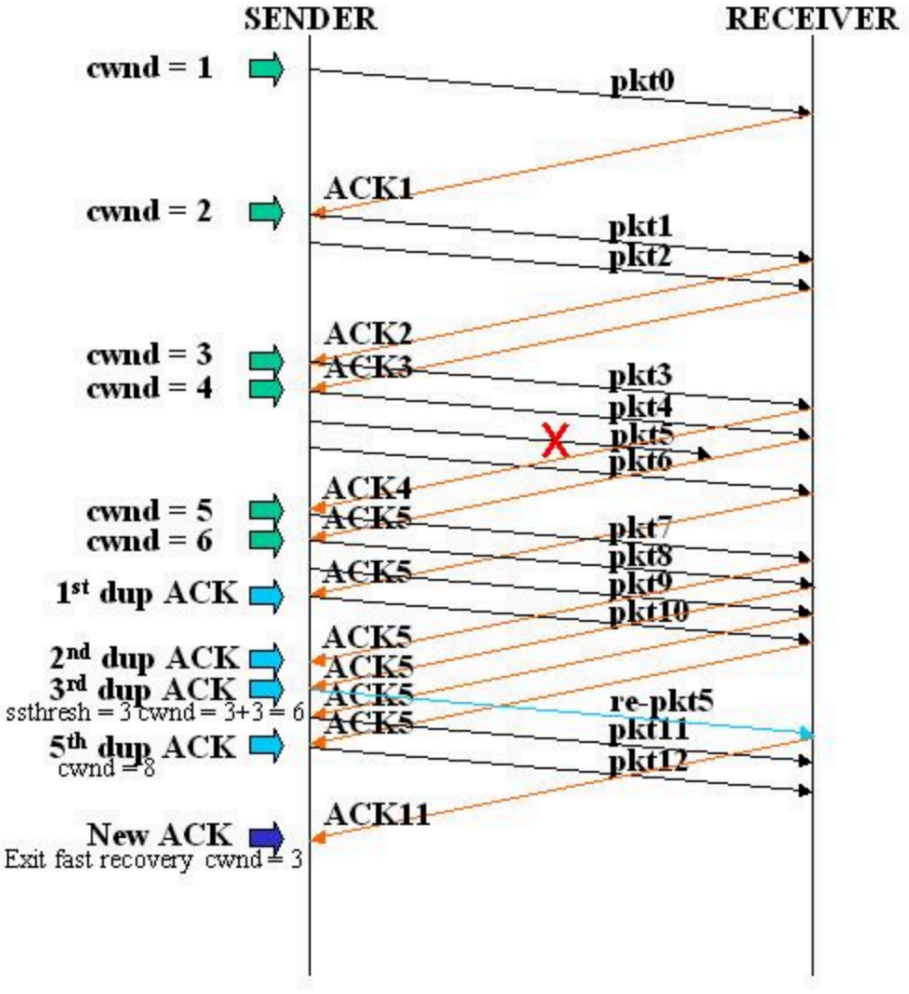


Source: https://www.youtube.com/watch?v=aUKE6WTTbH8

Fast Recovery

After receiving 3 duplicate ACKs in a row:

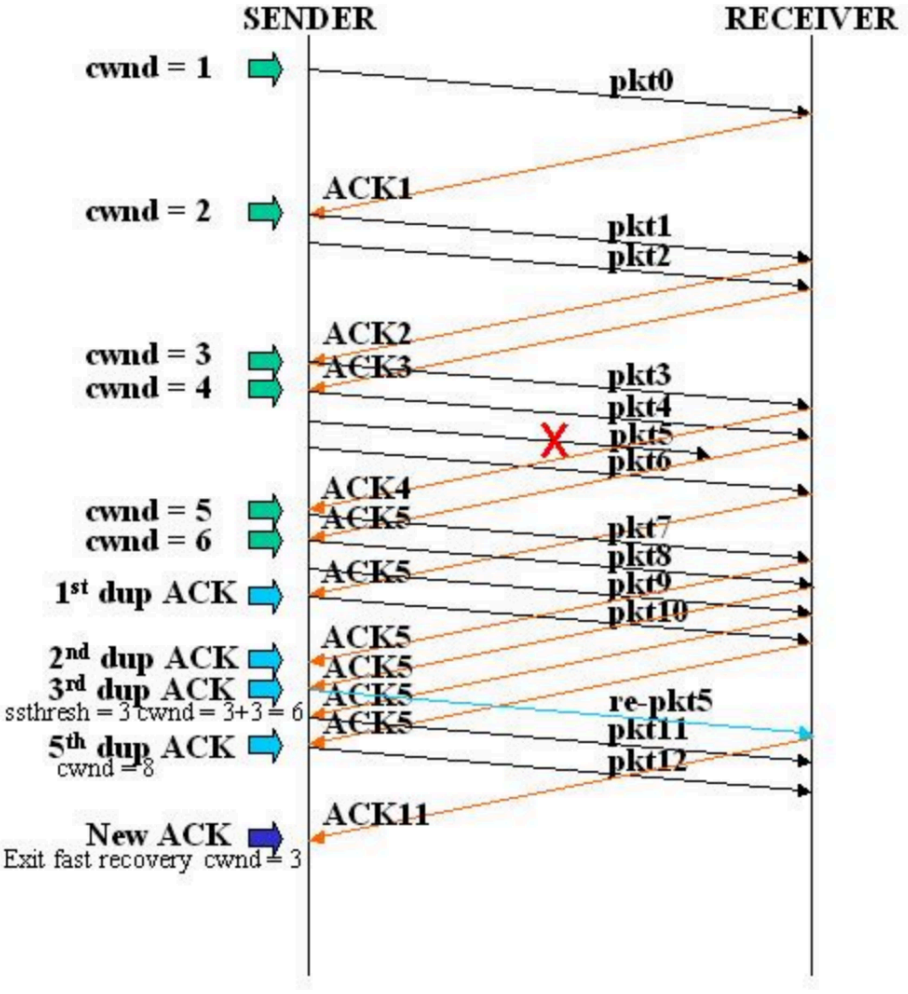
1) Set *ssthresh* = cwnd/2.
2) Retransmit the missing segment.
3) Set cwnd = ssthresh + 3.
4) Each time another duplicate ACK arrives, set cwnd = cwnd + 1. Then, send a new data segment if allowed by the value of cwnd.



Source: https://www.isi.edu/nsnam/DIRECTED_RESEARCH/DR_WANIDA/DR/JavisInActionFastRecoveryFrame.html

Fast Recovery

5) Once receive a new ACK (an ACK which acknowledges all intermediate segments sent between the lost packet and the receipt of the first duplicate ACK), exit fast recovery. This causes setting *cwnd* to *ssthresh* (the ssthresh in step 1). Then, continue with linear increasing due to congestion avoidance algorithm.

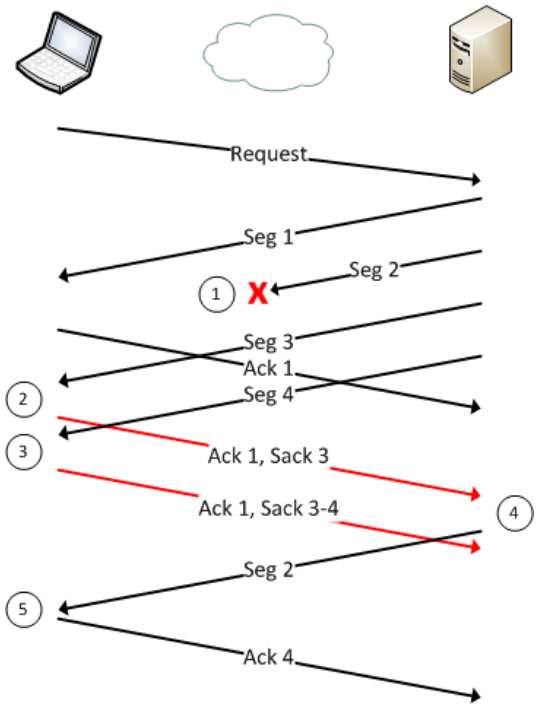


Selective Acknowledge (SACK)

- **Selective ACK** (SACK) option is implemented at the receiver.
- The sender does not have to guess from dupACKs what has gotten through.

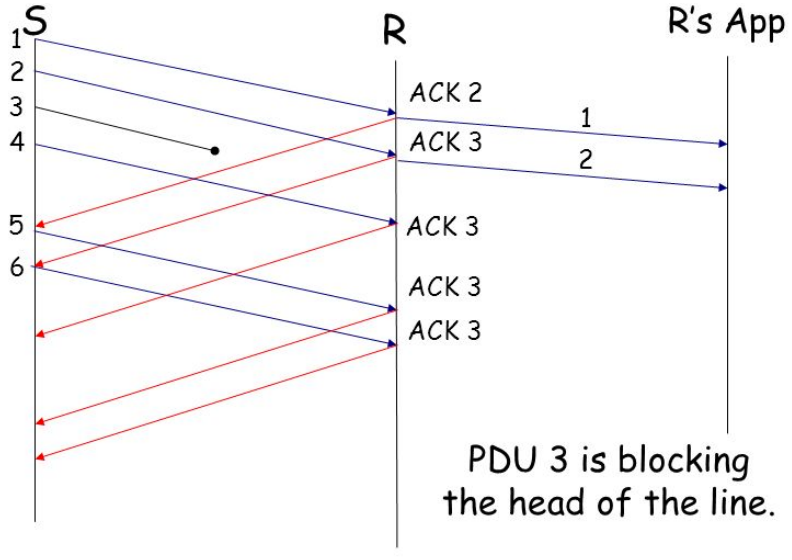
The receiver can send an ACK that says:

- All packets up through N have been received (the cumulative ACK)
- *All packets up through M have been received except for x, y, z.*



Source: http://packetlife.net/blog/2010/jun/17/tcp-selective-acknowledgments-sack/

Head-of-Line Blocking in TCP



Source: https://www.slideshare.net/shuyaosaki/introduction-to-quic

TCP vs. UDP

TCP	UDP
Keeps track of lost packets. Makes sure that lost packets are re-sent	Doesn’t keep track of lost packets
Adds sequence numbers to packets and reorders any packets that arrive in the wrong order	Doesn’t care about packet arrival order
Slower, because of all added additional functionality	Faster, because it lacks any extra features
Does not support multicast or broadcast	Support multicast and broadcast
Example services that use TCP: - HTTP - FTP	Example services that use UDP: - VoIP - DHCP

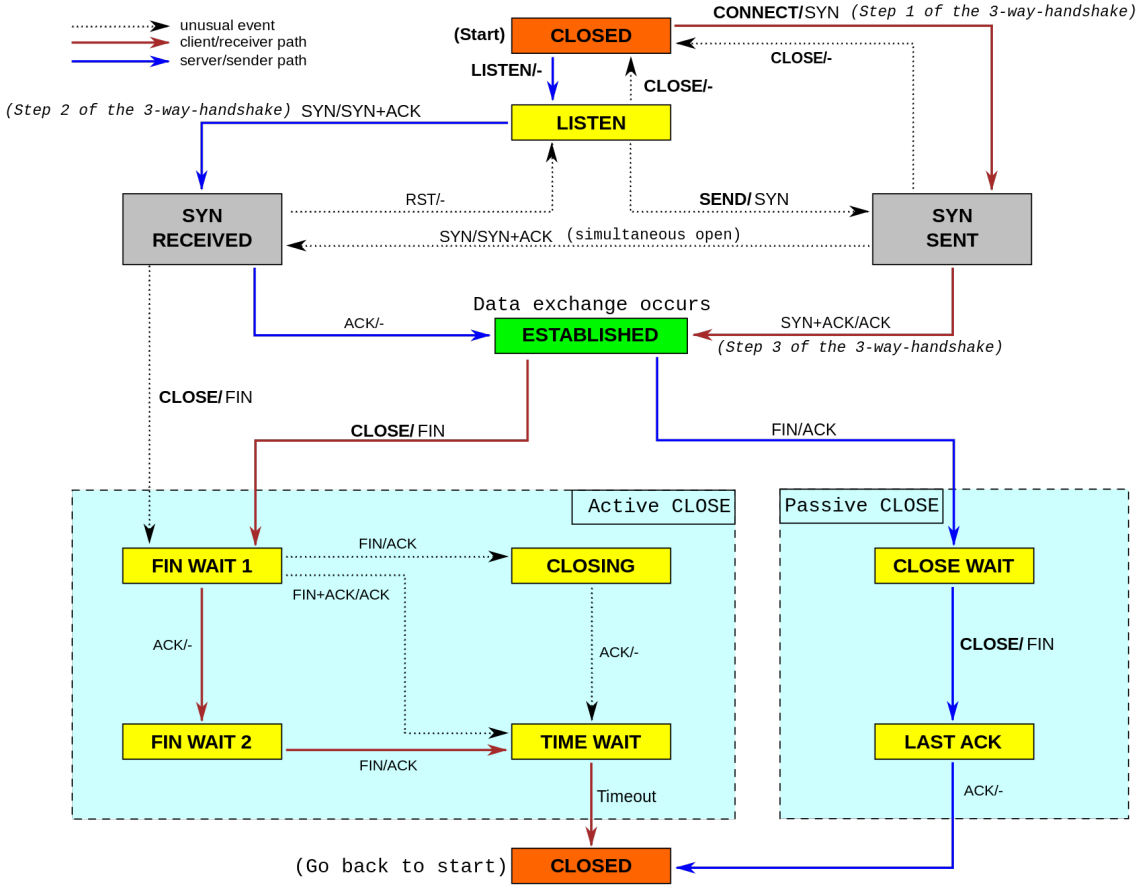
Reading List

Chapter 12-13

Types of Sockets

- **Datagram sockets:** connectionless sockets which use UDP
- **Stream sockets:** connection-oriented sockets which use TCP, SCTP or DCCP
- **Raw sockets:** the transport layer is bypassed and the packet headers are made accessible to the application. There is no port number in address, only IP address.

Simplified TCP State Diagram



State Machine