523454

Computer Network Programming

Lecture 2: TCP/IP Transport Layer Protocols

Dr. Parin Sornlertlamvanich,

parin.s@sut.ac.th

Protocols

- Devices communication needs a mechanism
  - Protocol

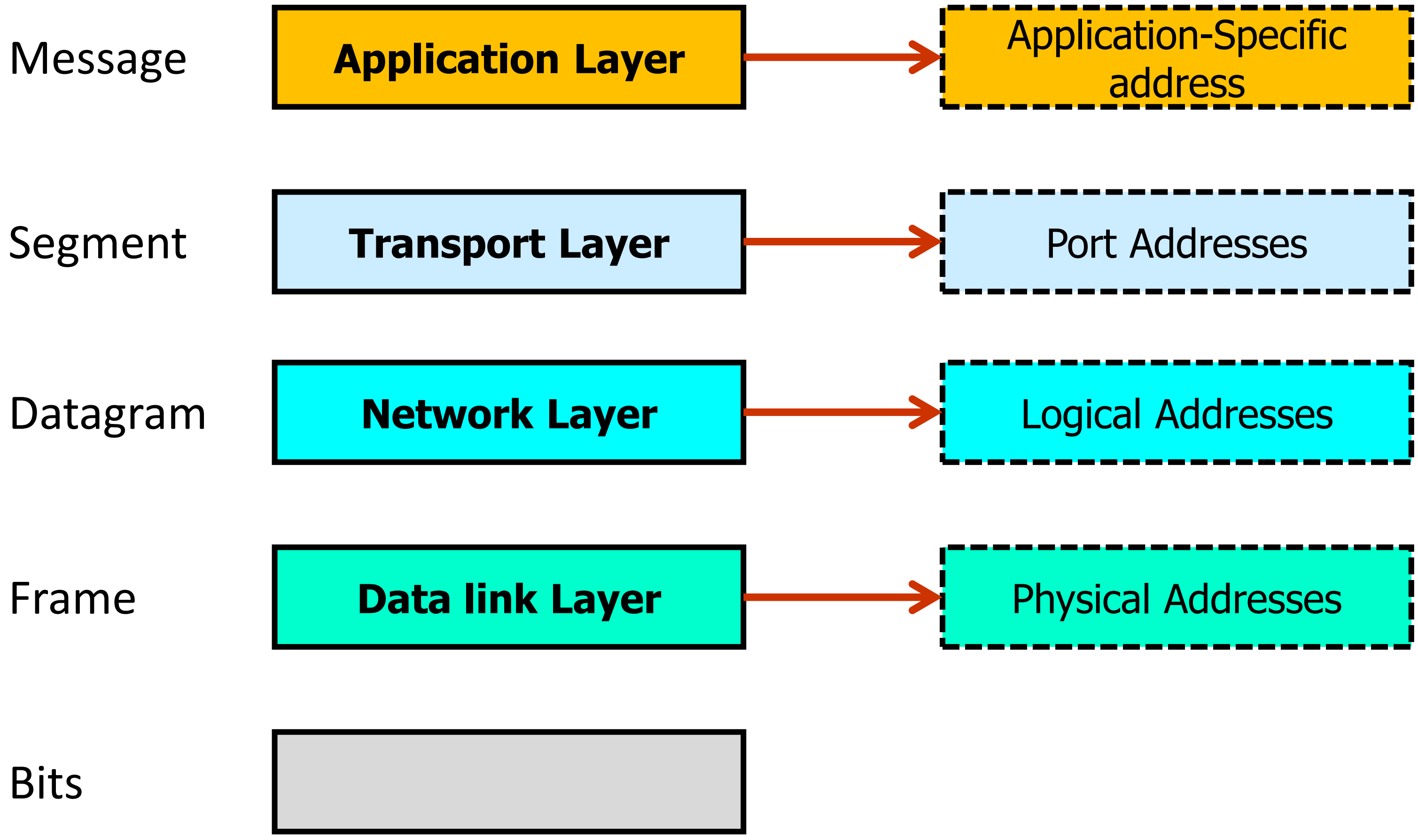
- A protocol for agreeing HOW to communicate
  - NOT concerned with WHAT is to be communicated
  - BUT different protocols for different communications

- What is data to one protocol is control information to another

Protocols (cont.)

- Rules for:
  - How to begin communication
  - How to carry on communication
    - Sequencing
  - Ending communication

Address in the TCP/IP Protocol suite



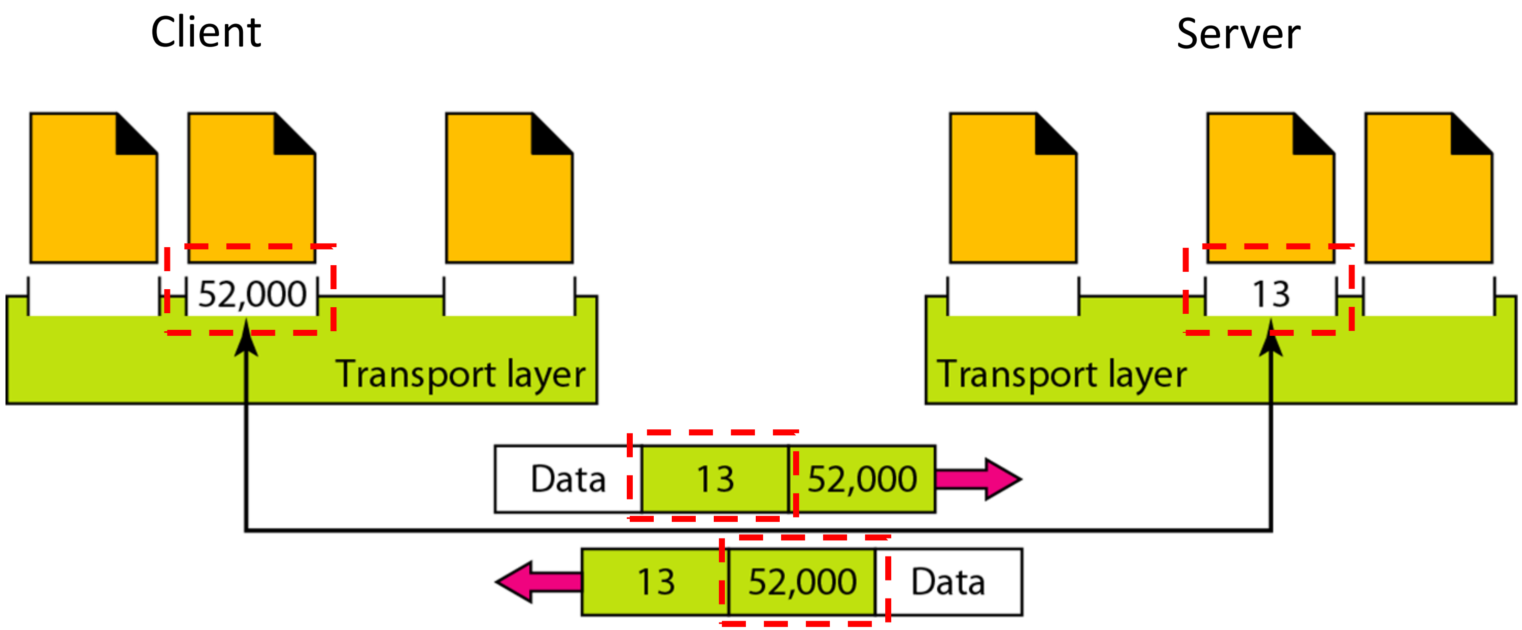
Transport Over IP

- The transport layer
  - Process-to-process delivery
  - The delivery of a packet

- Multiplexing and demultiplexing

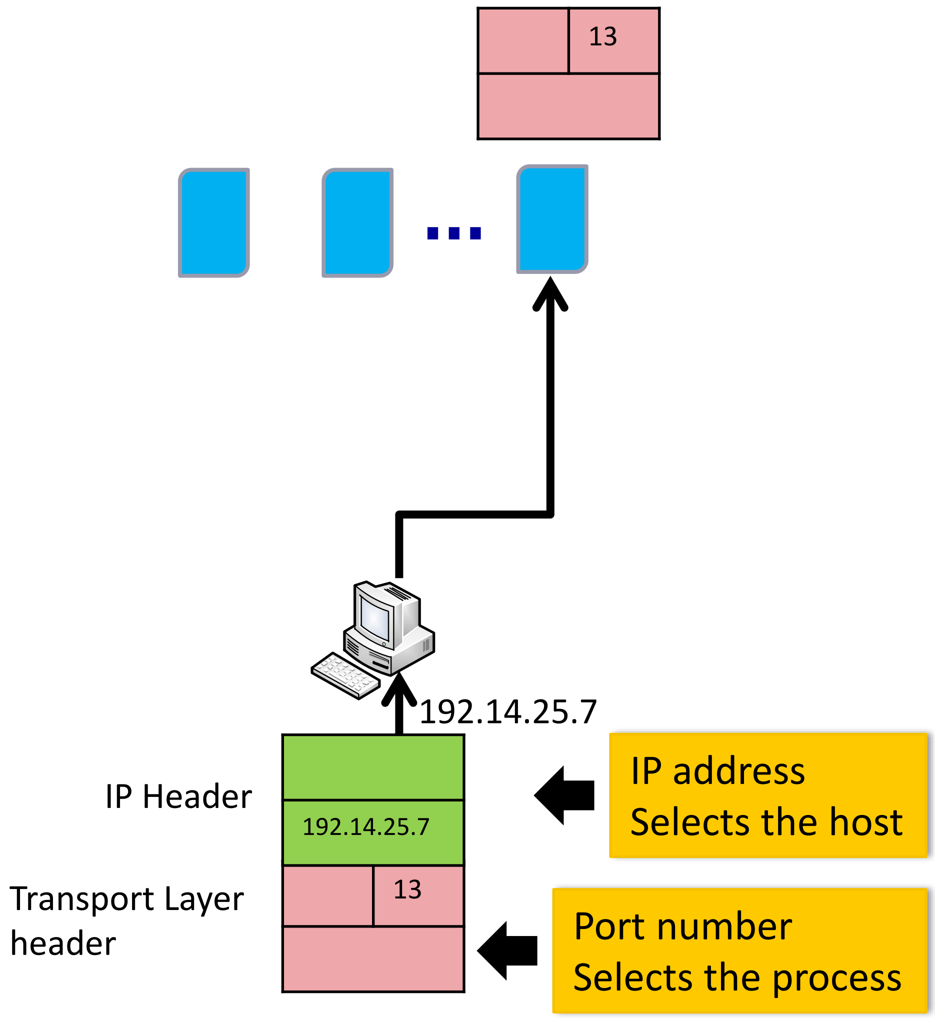
- Process-to-process communication
  - User Datagram Protocol (UDP)
  - Transmission Control protocol (TCP)

Port Numbers

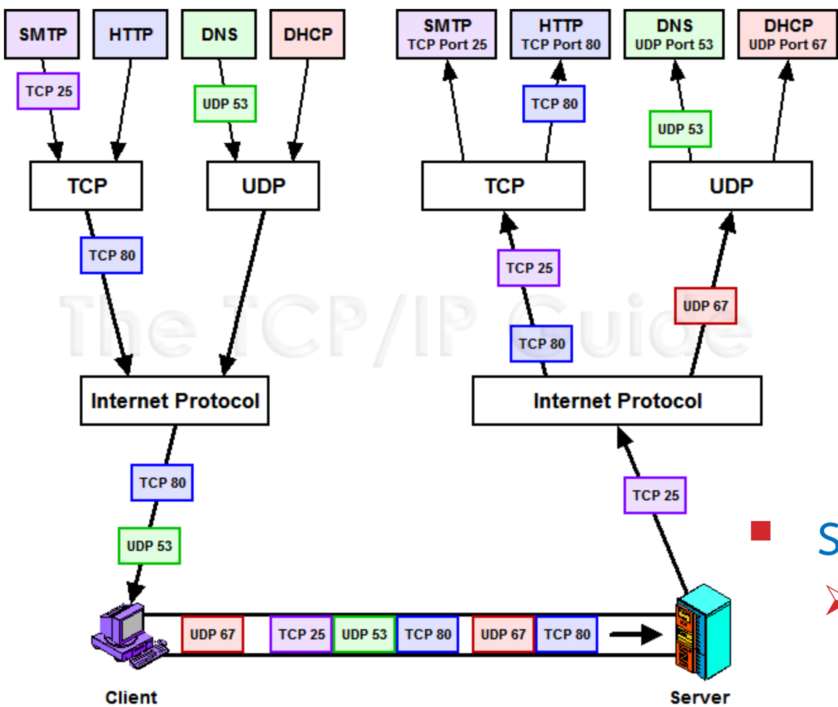


- At transport layer, we need a transport layer address
  - Port number
- Well-known port numbers
- Example, the server process must use the well-known port number 13

IP addresses and port numbers



Process Multiplexing/Demultiplexing Using TCP/UDP Ports



TCP Segment Header Format

Bit #	0 – 7	8 – 15	16 – 23	24 – 31
0	Source Port		Destination Port	
32	Sequence Number			
64	Acknowledgment Number			
96	Data Offset / Res	Flags	Window Size	
128	Header and Data Checksum		Urgent Pointer	
160...	Options			

UDP Datagram Header Format

Bit #	0 – 7	8 – 15	16 – 23	24 – 31
0	Source Port		Destination Port	
32	Length		Header and Data Checksum	

- Source Port and Destination Port
  - Identify the originating process on the source machine, and the destination process on the destination machine
    - from 0 to 65,535
    - both UDP and TCP use the same range of port numbers
  - Protocol field that specifies whether it is carrying a TCP message or a UDP message

Transport vs. Network layer

- *Network layer:* logical communication between hosts
- *Transport layer:* logical communication between processes

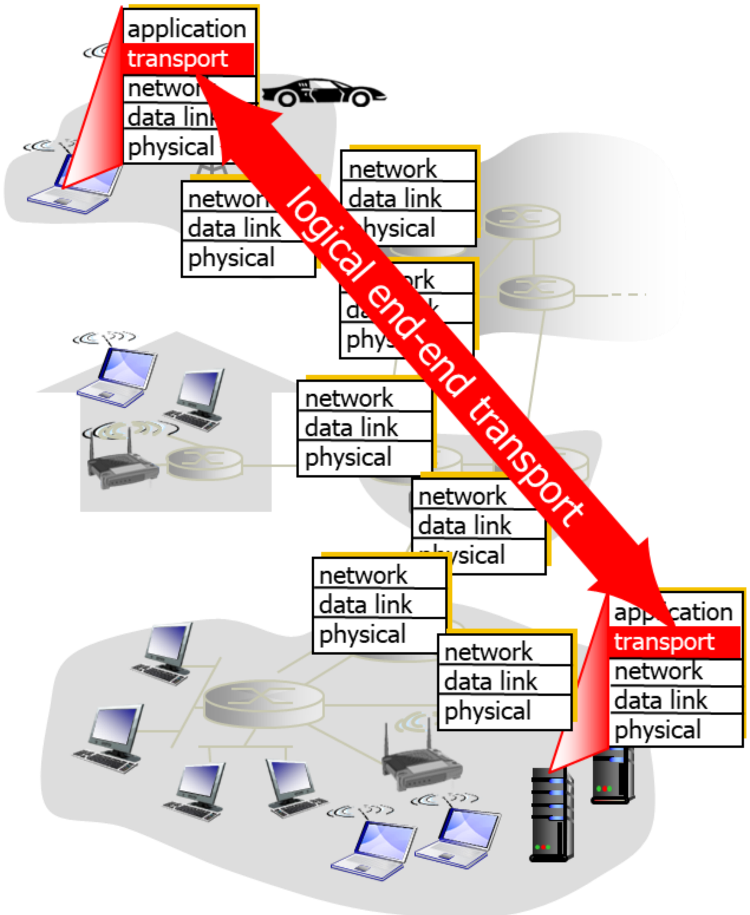
Household analogy:

12 kids in Ann's house sending letters to 12 kids in Bill's house:

- Hosts = houses
- Processes = kids
- App messages = letters in envelopes
- Transport protocol = Ann and Bill who demux to in-house siblings
- Network-layer protocol = postal service

Internet Transport-layer protocols

- Reliable, in-order delivery (TCP)
  - congestion control
  - flow control
  - connection setup
- Unreliable, unordered delivery: UDP
  - No-frills extension of “best-effort” IP
- Services not available:
  - delay guarantees
  - bandwidth guarantees



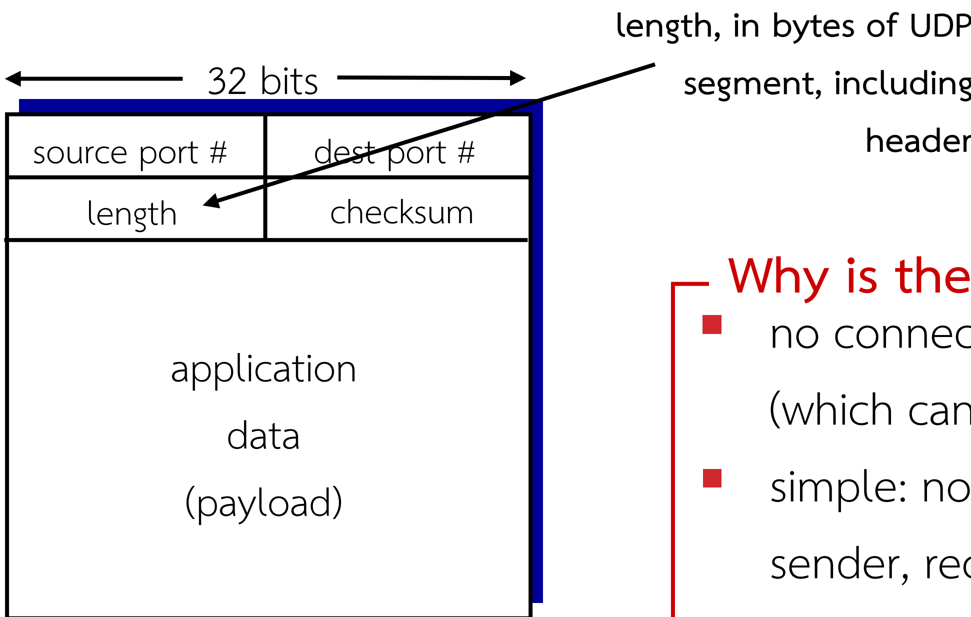
UDP: User Datagram Protocol [RFC 768]

- "No frills," "bare bones" Internet transport protocol
- "Best effort" service, UDP segments may be:
  - lost
  - delivered out-of-order to app
- Connectionless:
  - no handshaking between UDP sender, receiver
  - each UDP segment handled independently of others

- UDP use:
  - streaming multimedia apps (loss tolerant, rate sensitive)
  - DNS
  - SNMP

- Reliable transfer over UDP:
  - add reliability at application layer
  - application-specific error recovery!

UDP: Segment Header



UDP segment format

Why is there a UDP?

- no connection establishment (which can add delay)
- simple: no connection state at sender, receiver
- small header size
- no congestion control: UDP can blast away as fast as desired

UDP checksum

Goal: detect "errors" (e.g., flipped bits) in transmitted segment

Sender:

- Treat segment contents, including header fields, as sequence of 16-bit integers
- Checksum: addition of segment contents (padding = set to zero)
- Sender puts checksum value into UDP checksum field

Receiver:

- Compute checksum of received segment
- Check if computed checksum equals checksum field value:
  - NO - error detected
  - YES - no error detected. But maybe errors nonetheless

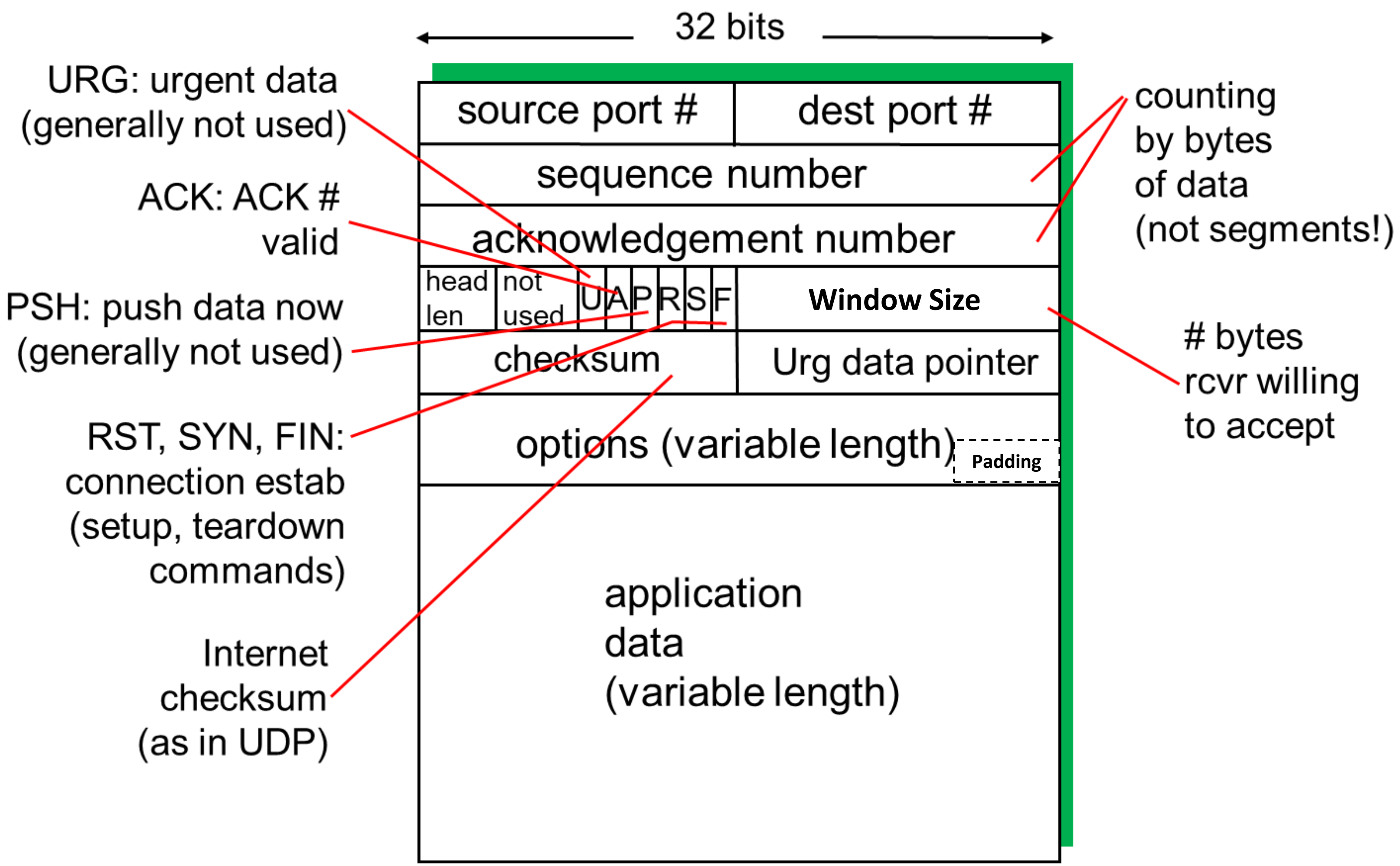
UDP Checksum calculation

10011001 00010010	→ 153.18
00001000 01101001	→ 8.105
10101011 00000010	→ 171.2
00001110 00001010	→ 14.10
00000000 00010001	→ 0 and 17
00000000 00001111	→ 15
00000100 00111111	→ 1087
00000000 00001101	→ 13
00000000 00001111	→ 15
00000000 00000000	→ 0 (checksum)
01010100 01000101	→ T and E
01010011 01010100	→ S and T
01001001 01001110	→ I and N
01000111 00000000	→ G and 0 (padding)
10010110 11101011	→ Sum
01101001 00010100	→ Checksum

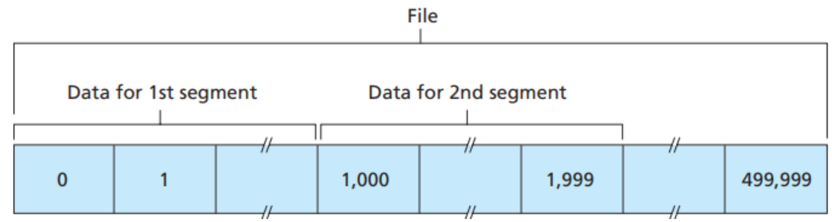
TCP: Overview RFCs: 793,1122,1323, 2018, 2581

- Point-to-Point:
  - One sender, One receiver
- Reliable, in-order byte steam:
  - 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

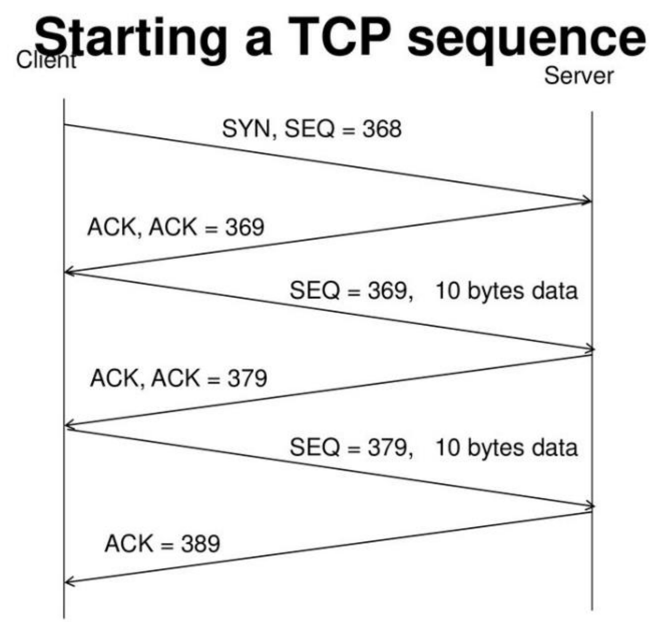
TCP segment structure



TCP seq. numbers, ACKs

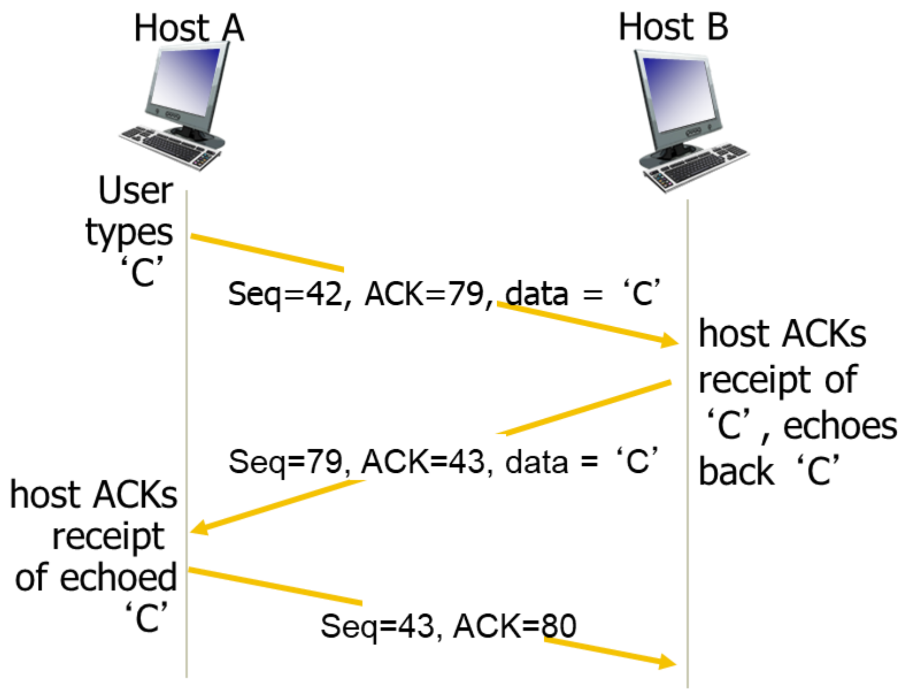


- Sequence numbers:
  - ➢ Byte stream "number" of first byte in segment's data



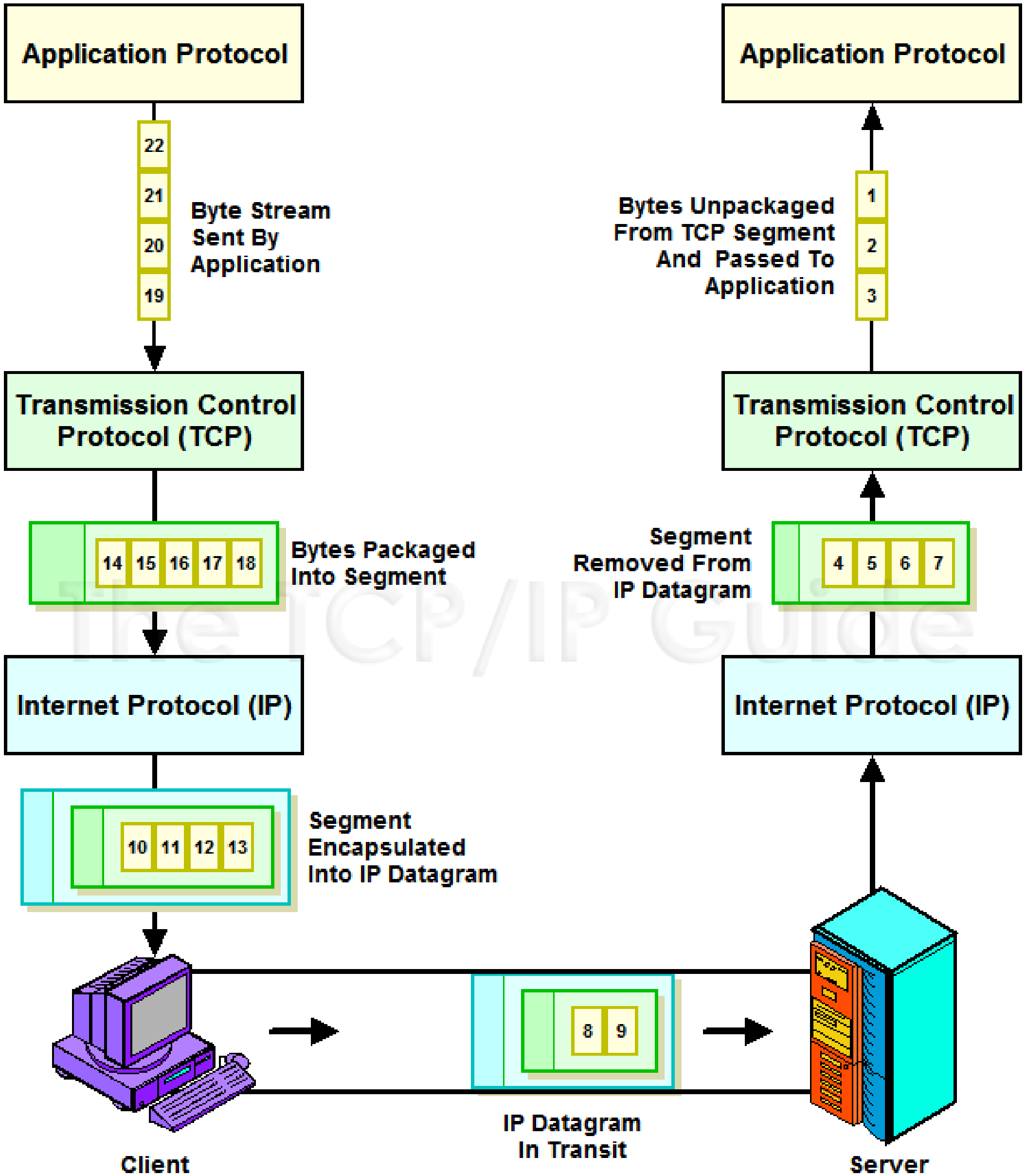
- Acknowledgements:
  - ➢ Seq # of next byte expected from other side
  - ➢ Cumulative ACK
- How receiver handles out-of-order segments
  - ➢ Ans: TCP spec doesn't say, - up to implementor

TCP seq. numbers, ACKs



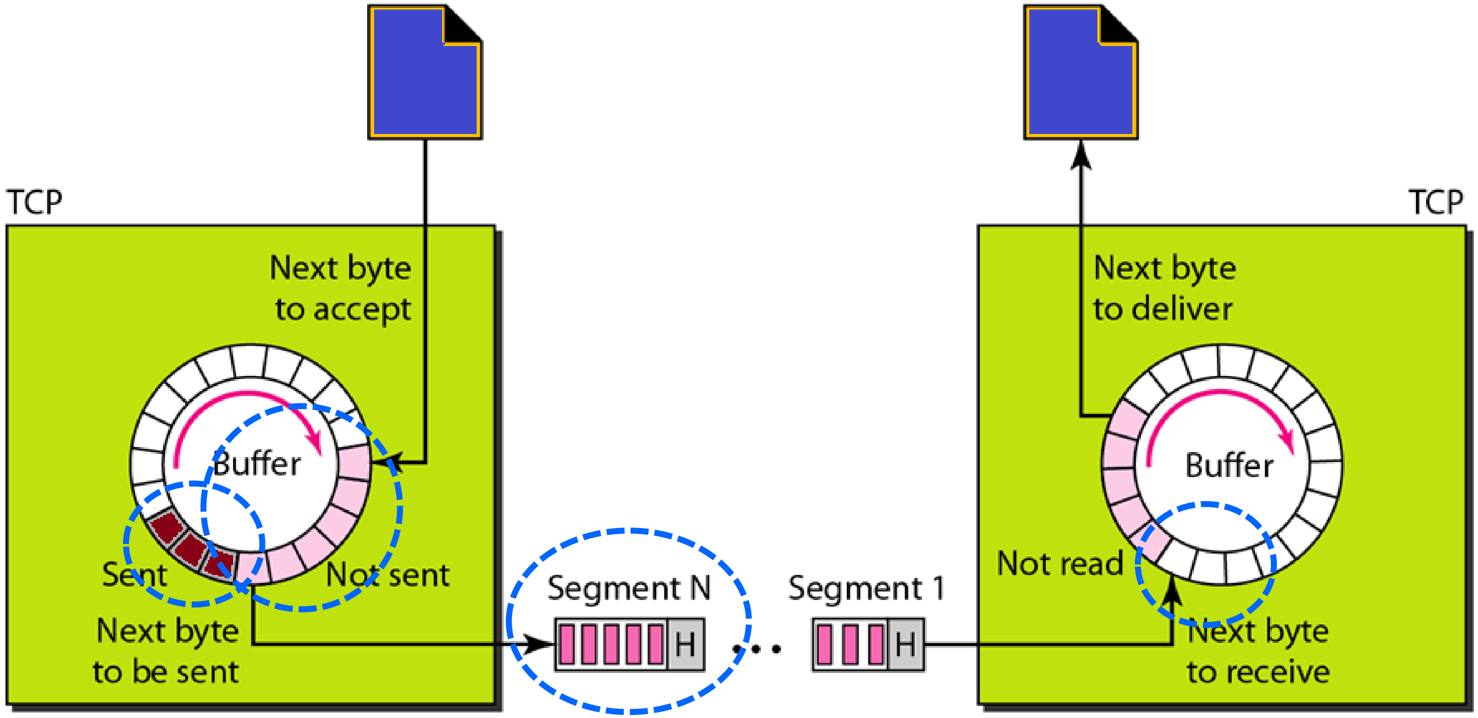
Simple telnet scenario

TCP Data Stream Processing



- TCP is designed to have applications send data to it as a stream of bytes, rather than requiring fixed-size messages to be used.
- This provide maximum flexibility for a wide variety of uses, because applications don't need to worry about data packaging, and can send files or messages of any size.

TCP Segments

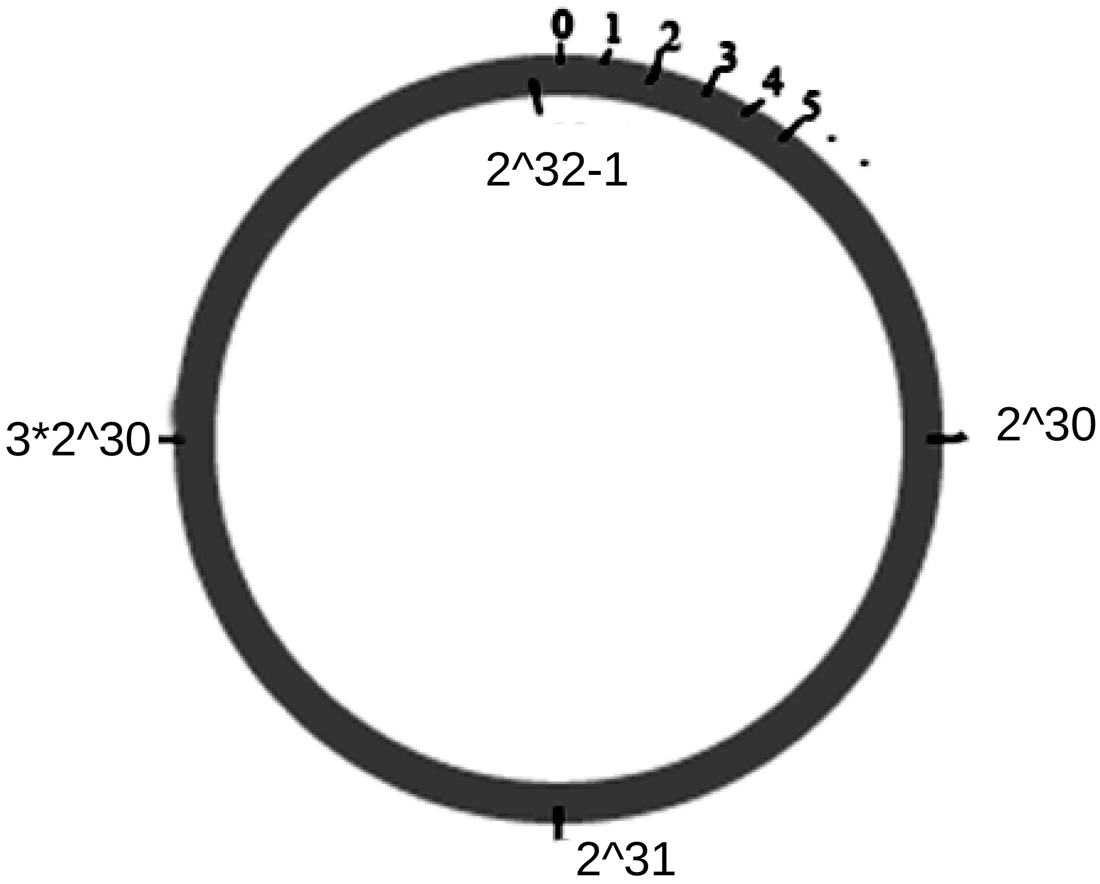


TCP Sequenced Data

Source Port								Destination Port
Sequence Number								
Acknowledgment Number								
HdrLen	Reserved	U	A	P	R	S	F	Window Size
Checksum								Urgent Pointer
Data	Data							

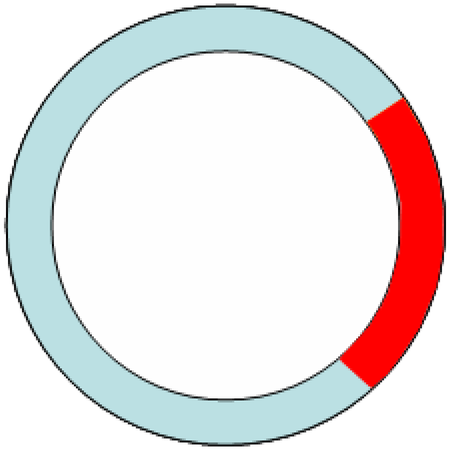
- Sequence number in header identifies the first data byte in the segment
- Sequence numbers of later data bytes obtained by arithmetic

TCP Sequence Number Space



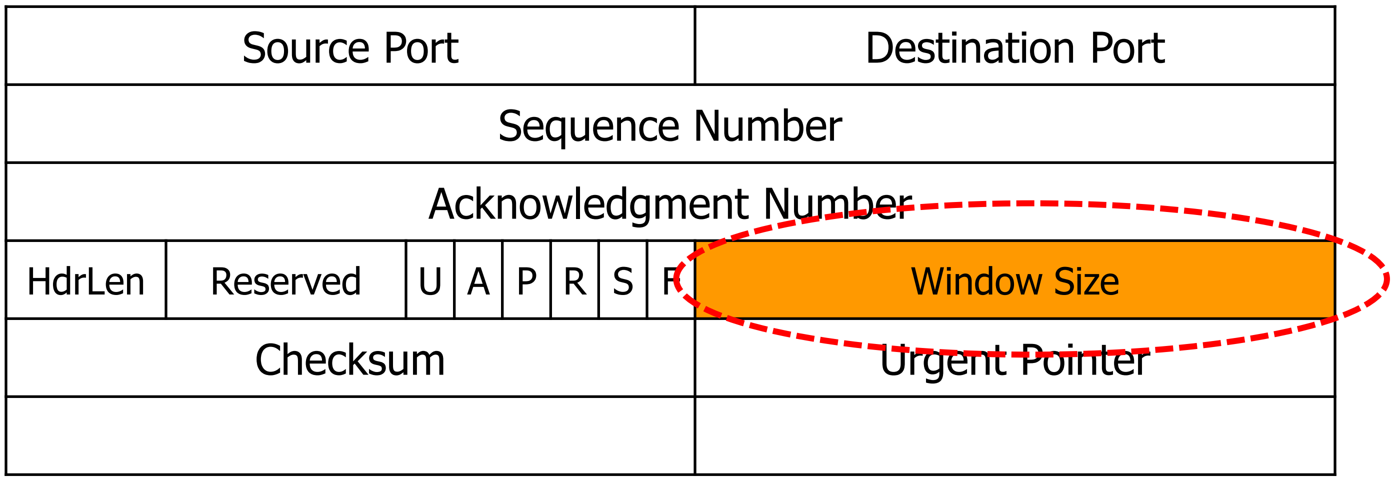
- Sequence numbers run from 0 to 2^32 - 1
- 0, 1, 2, 3, ..., 2^32-2, 2^32-1, 0, 1, 2, ...

TCP Window



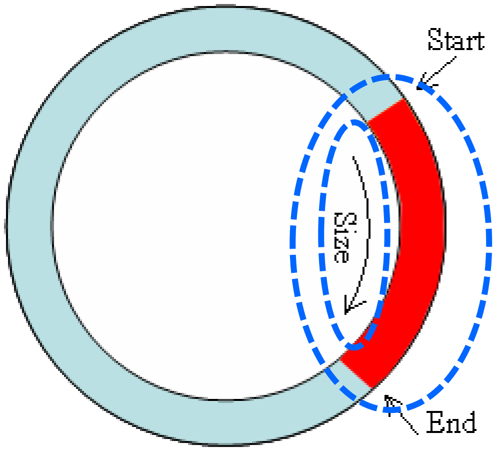
- A segment of the sequence number space
- Data can only be transmitted in the window
- Allows receiver to control buffer space consumed

TCP Window Size



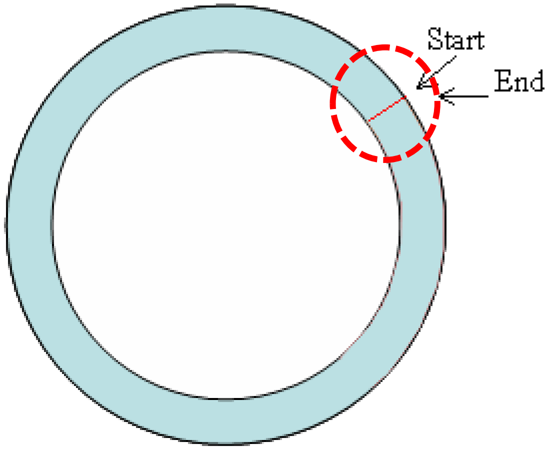
- Window Size sent from receiver to sender of data
- Indicates the current window size available

TCP Window Size (cont.)



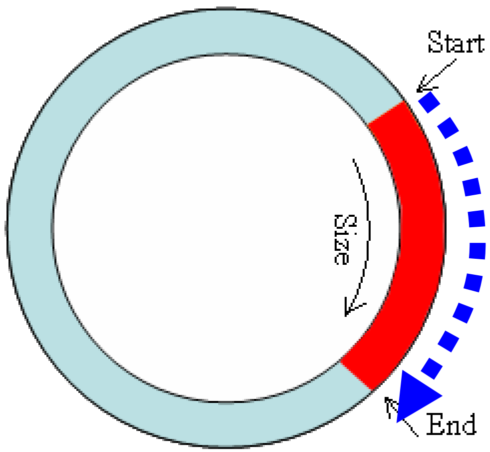
- Window Size
  - the current size of the buffer allocated to accept data for this connection
- End calculated from Start + Size (-1)

A better representation (scaling)



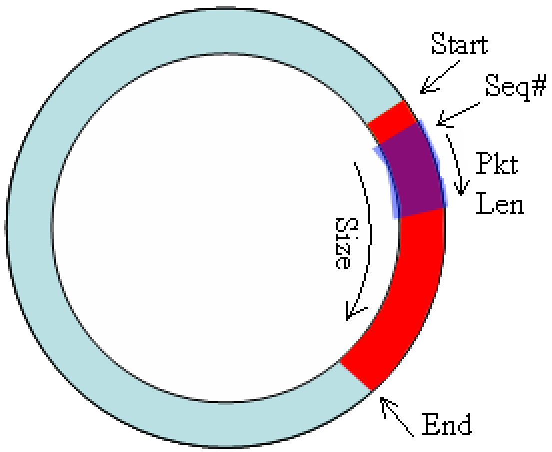
- 2^16 bytes max window size (65,536)
  - Max number of bytes that a receiver can accept
- 2^32 bytes sequence space (4,294,967,296)
- Max window approx 1/2 of 1/100 of 1 degree

Filling the Window



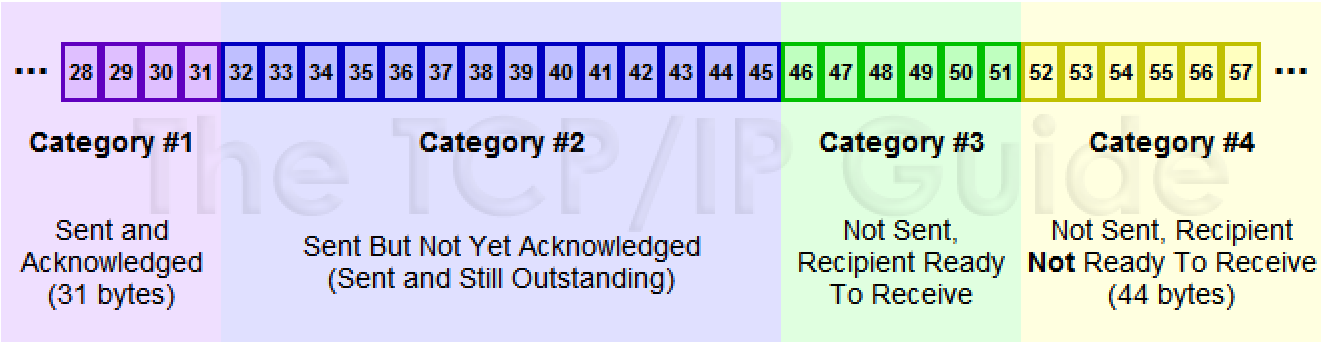
- Data transmitted gradually fills the window

Sending data



- TCP sequence number, and length of data
  - IP length - IP header len - TCP header len

TCP's Stream-Oriented Sliding Window

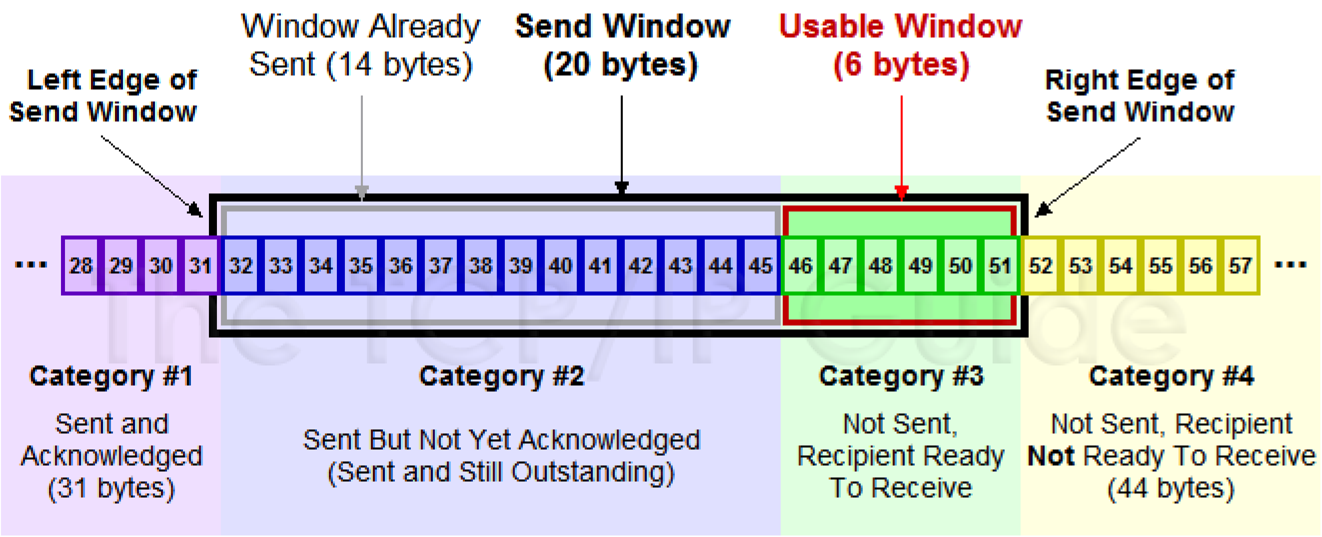


TCP Transmission Stream Into Categories

- Bytes Sent And Acknowledged
  - The earliest bytes in the stream will have been sent and acknowledged (31 bytes)
- Bytes Sent But Not Yet Acknowledged
  - These are the bytes that the device has sent but for which it has not yet received an acknowledgment (14 bytes)
- Bytes Not Yet Sent For Which Recipient Is Ready
  - These are bytes that have not yet been sent, but which the recipient has room for based on its most recent communication (6 bytes)
- Bytes Not Yet Sent For Which Recipient Is Not Ready
  - These are the bytes further “down the stream” which the sender is not yet allowed to send because the receiver is not ready (44 bytes)

http://www.tcpipguide.com/free/t_TCPSlidingWindowAcknowledgmentSystemForDataTranspo-5.htm

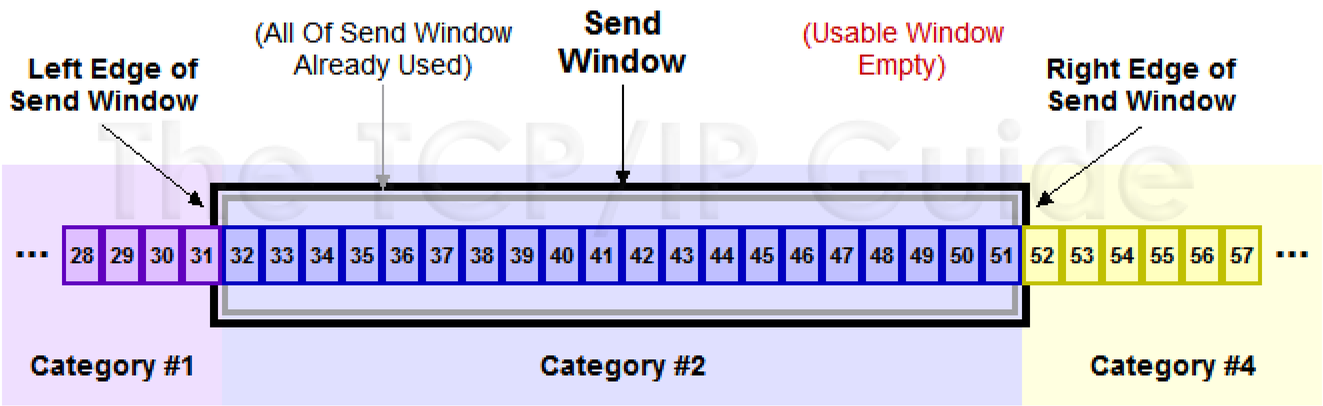
The Send Window and Usable Window



- Send Window (or Window): the number of bytes that the recipient is allowing the transmitter to have unacknowledged at one time
  - How many bytes the sender is allowed to transmit
  - The number of bytes for Category#2 + Category#3
  - The total window size is 20
- Usable window is defined as the amount of data the transmitter is still allowed to send

http://www.tcpipguide.com/free/t_TCPSlidingWindowAcknowledgmentSystemForDataTranspo-5.htm

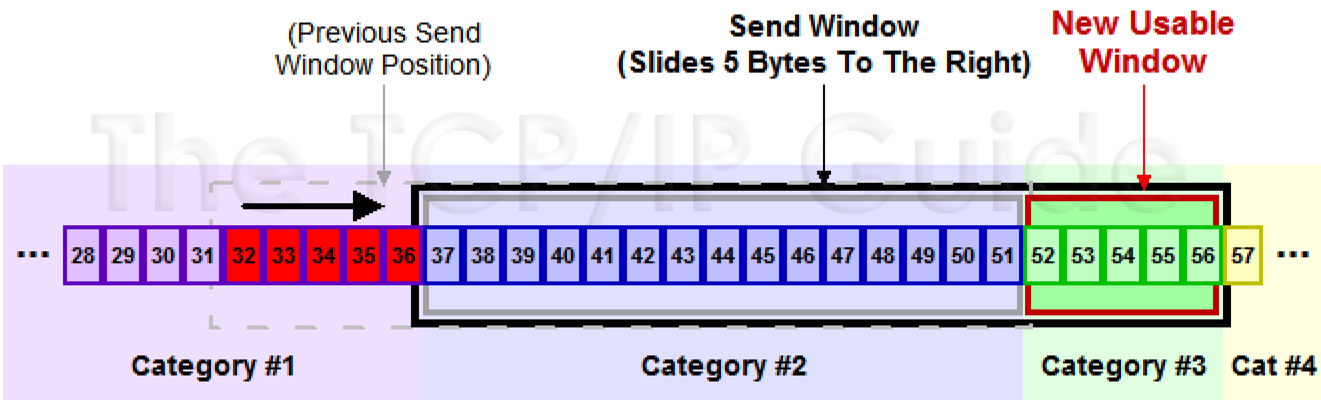
Changes to TCP Categories and Window Sizes



- Sender sends the 6 bytes in Cat#3
  - the 6 bytes will shift from Cat#3 to Cat#2
  - Cat#3 = None
- The usable window is now zero, and will remain so until an acknowledgment is received for bytes in Cat#2

http://www.tcpipguide.com/free/t_TCPSlidingWindowAcknowledgmentSystemForDataTranspo-5.htm

Processing Acks and Sliding the Send Window



- Sender sends 4 segments carried bytes 32 to 34, 35 to 36, 37 to 41 and 42 to 45 respectively
- The first, second and fourth segments arrived, but the <u>third did not</u>
  - The receiver will send back an acknowledgment only for bytes 32 to 36
- The receiver will hold bytes 42 to 45
  - This is necessary because TCP is a *cumulative acknowledgment system*
- When the sender receives this acknowledgment
  - the bytes from Cat#2 to Cat#1
  - “ TCP sliding window acknowledgment system”

Dealing With Missing Acknowledgments

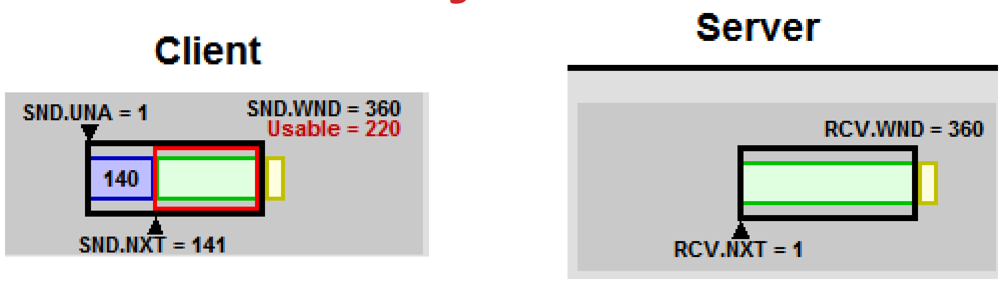
- **What about bytes 42 through 45???**
  - until segment#3 (containing bytes 37 to 41) shows up, the receiver will not send an acknowledgment

- **TCP device will re-send the lost segment, and hopefully this time it will arrive again.**
  - Unfortunately, one drawback of TCP is that since it does not separately acknowledge segments, this may mean retransmitting other segments that actually were received by the recipient (such as the segment with bytes 42 to 45).

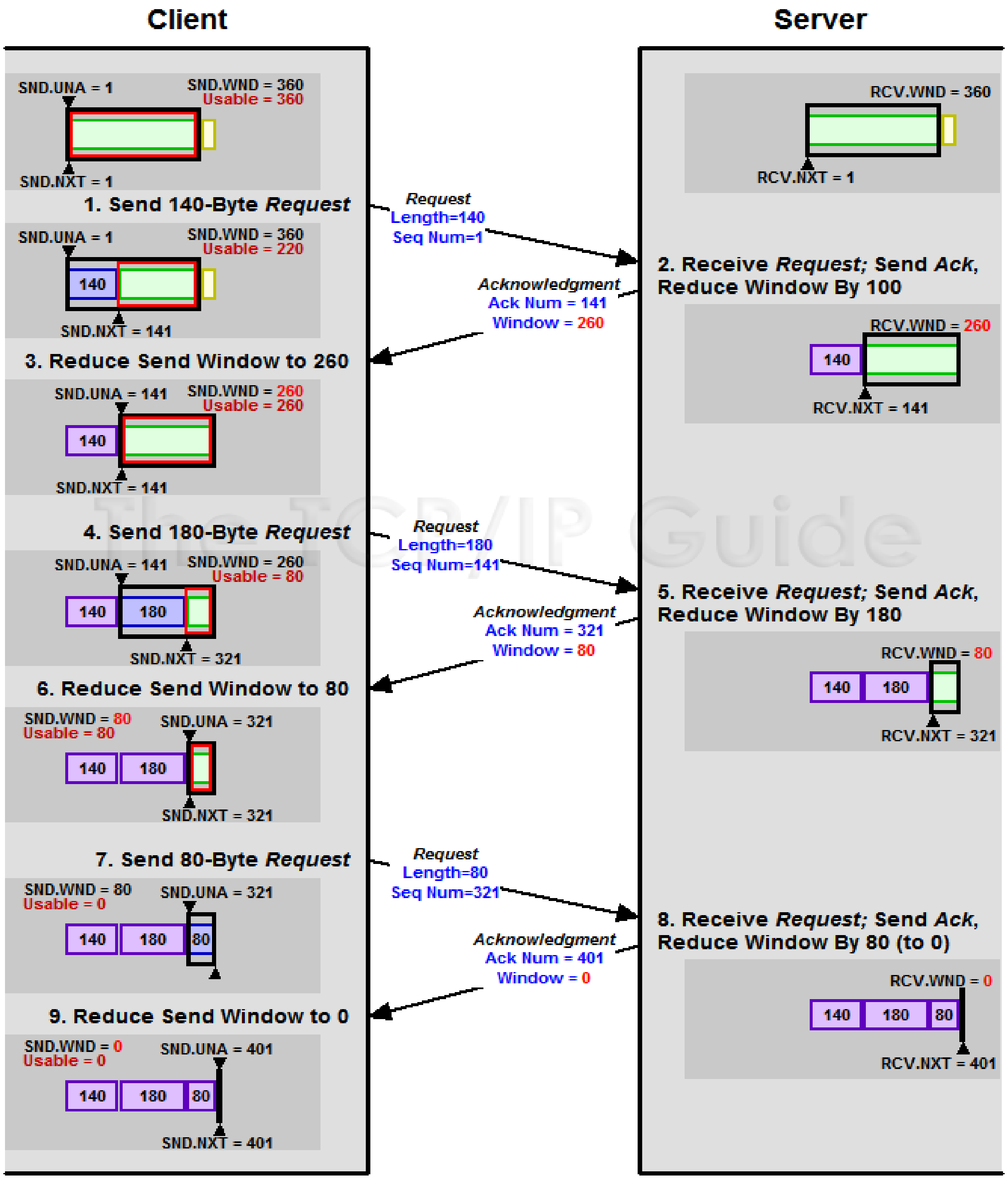
A Perfect Connection

- Sender picks sequence number
- Receiver sets window size
- Sender sends up to window size bytes of data
- Receiver acknowledges data and advances window
- Sender sends more data
    - until no more to send
- Receiver acknowledges final data
- Connection establishment & termination
    - Later…

TCP Window Size Adjustment



- Window size is the size of the device's receive buffer
- Window size represents how much data a device can handle from its peer at one time before it is passed to the application process
- "Ideal world"
  - 140 bytes go into the buffer, are acknowledged and immediately removed from the buffer
  - the buffer is of "infinite size" - the buffer's free space remains 360 bytes
- "Real world"
  - Server might be dealing with hundreds or even thousands of TCP connections
  - Server's TCP may not be able to immediately remove all 140 bytes from the buffer (the application might not be ready for the 140 bytes)
  - To change the window size that it advertises to the client, to reflect the fact that the buffer is partially filled



Client
Server
SND.UNA = 1
SND.WND = 360
Usable = 360
SND.NXT = 1
RCV.WND = 360
RCV.NXT = 1
1. Send 140-Byte *Request*
Request
Length=140
Seq Num=1
SND.UNA = 1
SND.WND = 360
Usable = 220
140
SND.NXT = 141
2. Receive *Request;* Send *Ack*, Reduce Window By 100
Acknowledgment
Ack Num = 141
Window = 260
RCV.WND = 260
140
RCV.NXT = 141
3. Reduce Send Window to 260
SND.UNA = 141
SND.WND = 260
Usable = 260
140
SND.NXT = 141
4. Send 180-Byte *Request*
Request
Length=180
Seq Num=141
SND.UNA = 141
SND.WND = 260
Usable = 80
140
180
SND.NXT = 321
5. Receive *Request;* Send *Ack*, Reduce Window By 180
Acknowledgment
Ack Num = 321
Window = 80
RCV.WND = 80
140
180
RCV.NXT = 321
6. Reduce Send Window to 80
SND.WND = 80
Usable = 80
SND.UNA = 321
140
180
SND.NXT = 321
7. Send 80-Byte *Request*
Request
Length=80
Seq Num=321
SND.WND = 80
Usable = 0
SND.UNA = 321
140
180
80
8. Receive *Request;* Send *Ack*, Reduce Window By 80 (to 0)
Acknowledgment
Ack Num = 401
Window = 0
RCV.WND = 0
140
180
80
RCV.NXT = 401
9. Reduce Send Window to 0
SND.WND = 0
Usable = 0
SND.UNA = 401
140
180
80
SND.NXT = 401
The TCP/IP Guide

TCP Notes

- TCP connections are bi-directional
  - Every packet has both sequence number & acknowledgement number
    - Sequence number indicates which data are in this packet (if any)
    - Acknowledgement indicates which data are expected to be received next
- Can acknowledge data received,
  - and send answer (any reply)
  - in the same packet
- Window size sent in every packet
  - Can be varied as buffer space at receiver varies

TCP round trip time, timeout

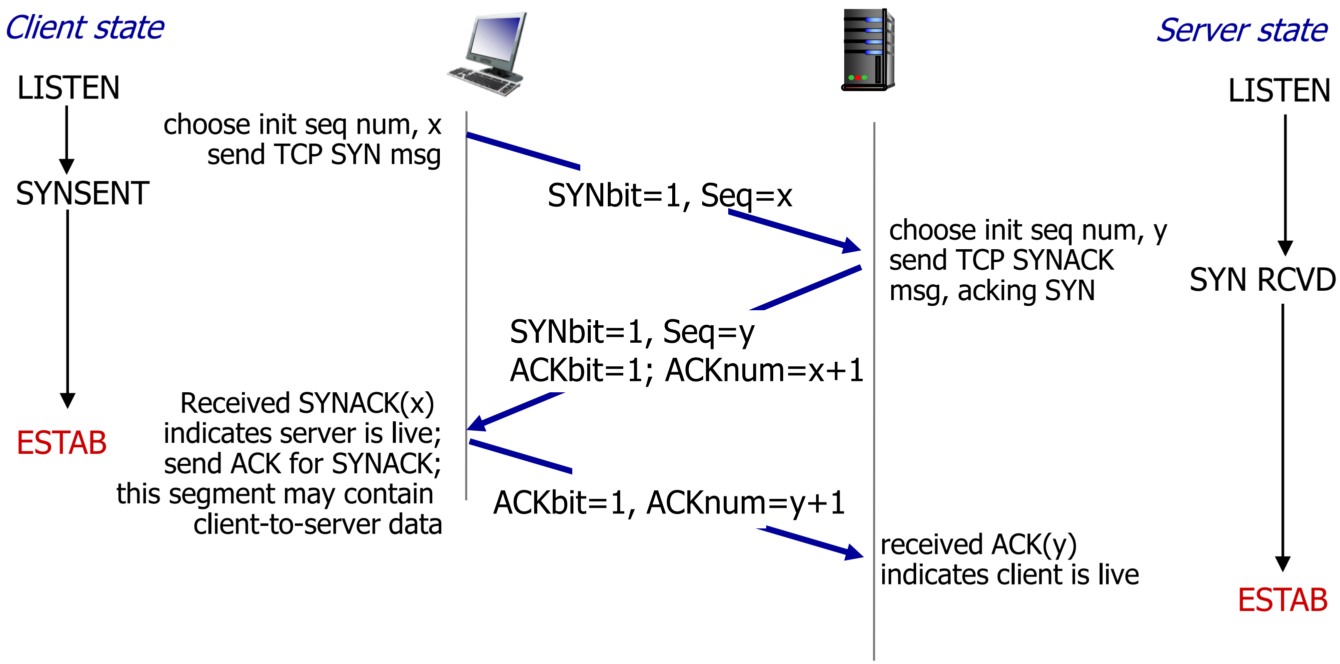
How to set TCP timeout value?

- longer than RTT
- but RTT varies
  - too short: premature timeout, unnecessary retransmissions
  - too long: slow reaction to segment loss

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

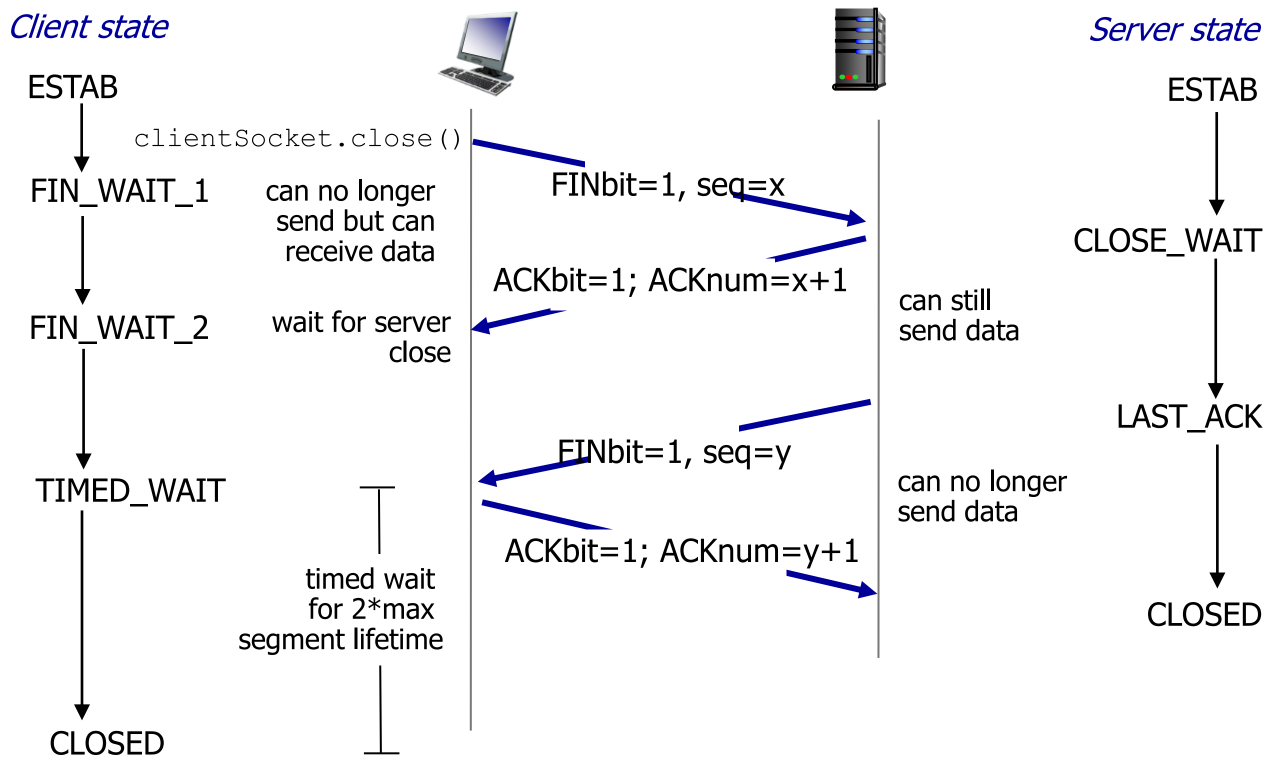
TCP 3-way handshake



TCP: closing a connection

- Client, Server each close their side of connection
  - Sending 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

TCP: closing a connection

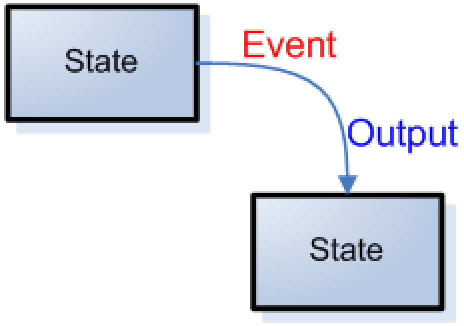


Finite State Machine (FSM)

- Using FSM to Explain Complex Protocols
  - More formal method of speificiation

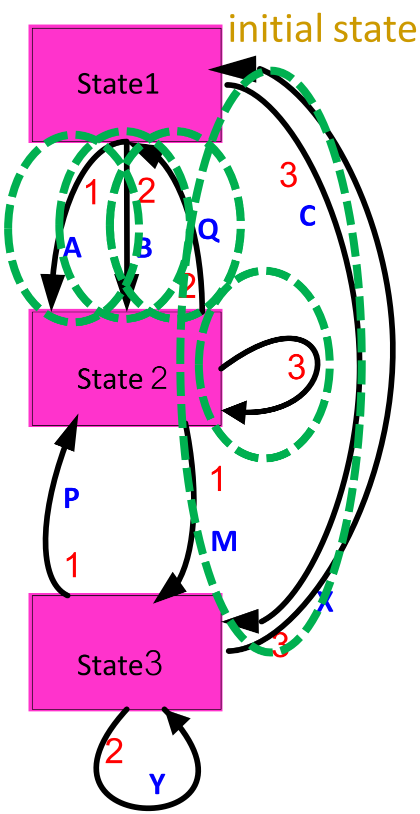
- Four essential conceptsspecific
  - State: The particular “status” that describes the protocol software on a machine at a given time.
  - Transition: The act of moving from one state to another.
  - Event: Something that causes a transition to occur between states.
  - Action (or Output): Something a device does in response to an event before it transitions to another state.

FSM Basics



- An event occurs
  - drawn beside a line
  - shows transition to a new state
- Some output may accompany transition
- Transition may return to the same state
  - Or may transition to another state

FSM (example)



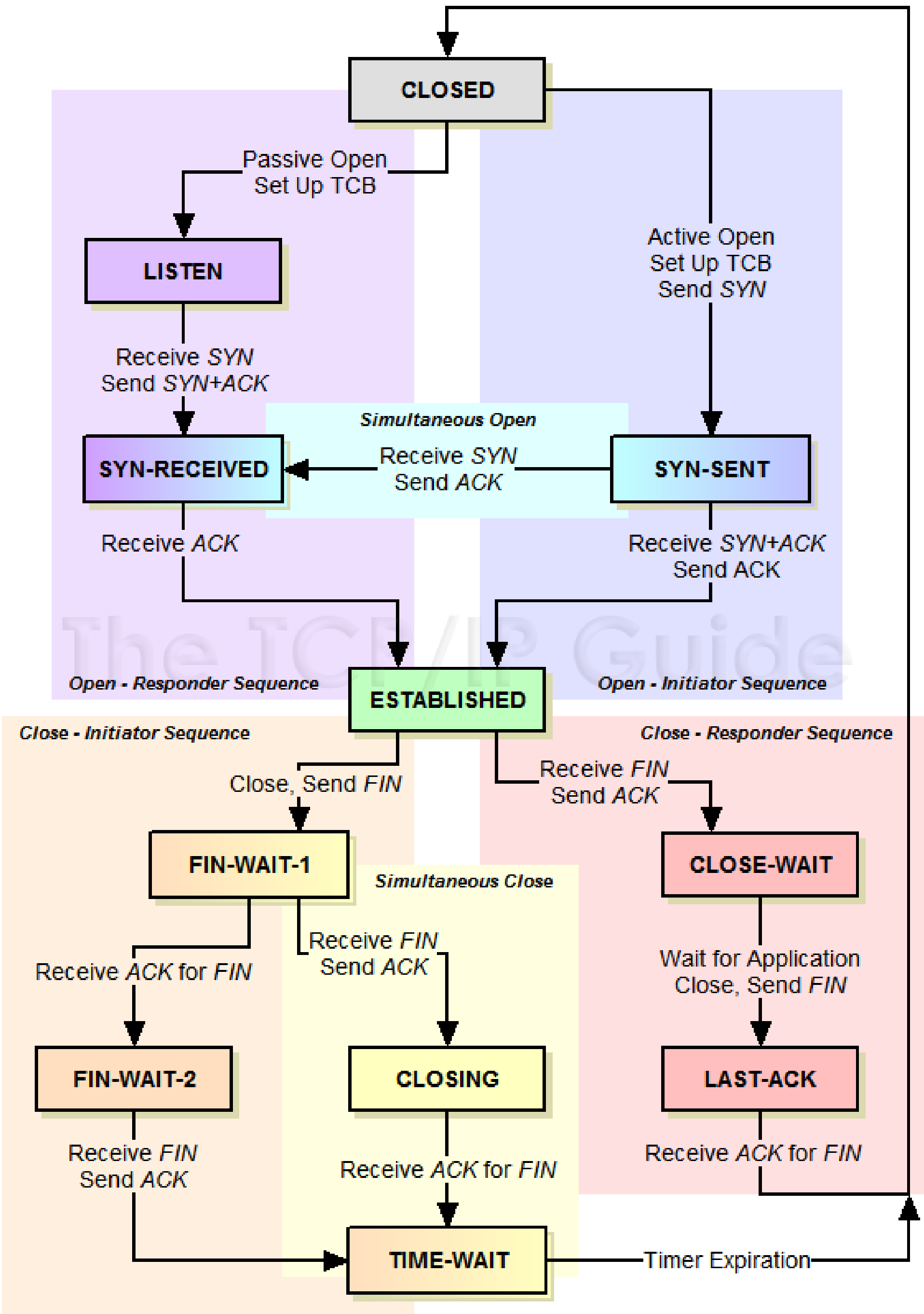
- Three States 1 2 and 3
- Three events that occur 1 2 and 3
- Several output actions
- State 1 is the initial state
- For input events

1 2 2 3 2 3

What states does FSM pass through?
What output actions are performed?

1 2 1 2 2 1 3

A Q B - Q C



TCP Finite State Machine

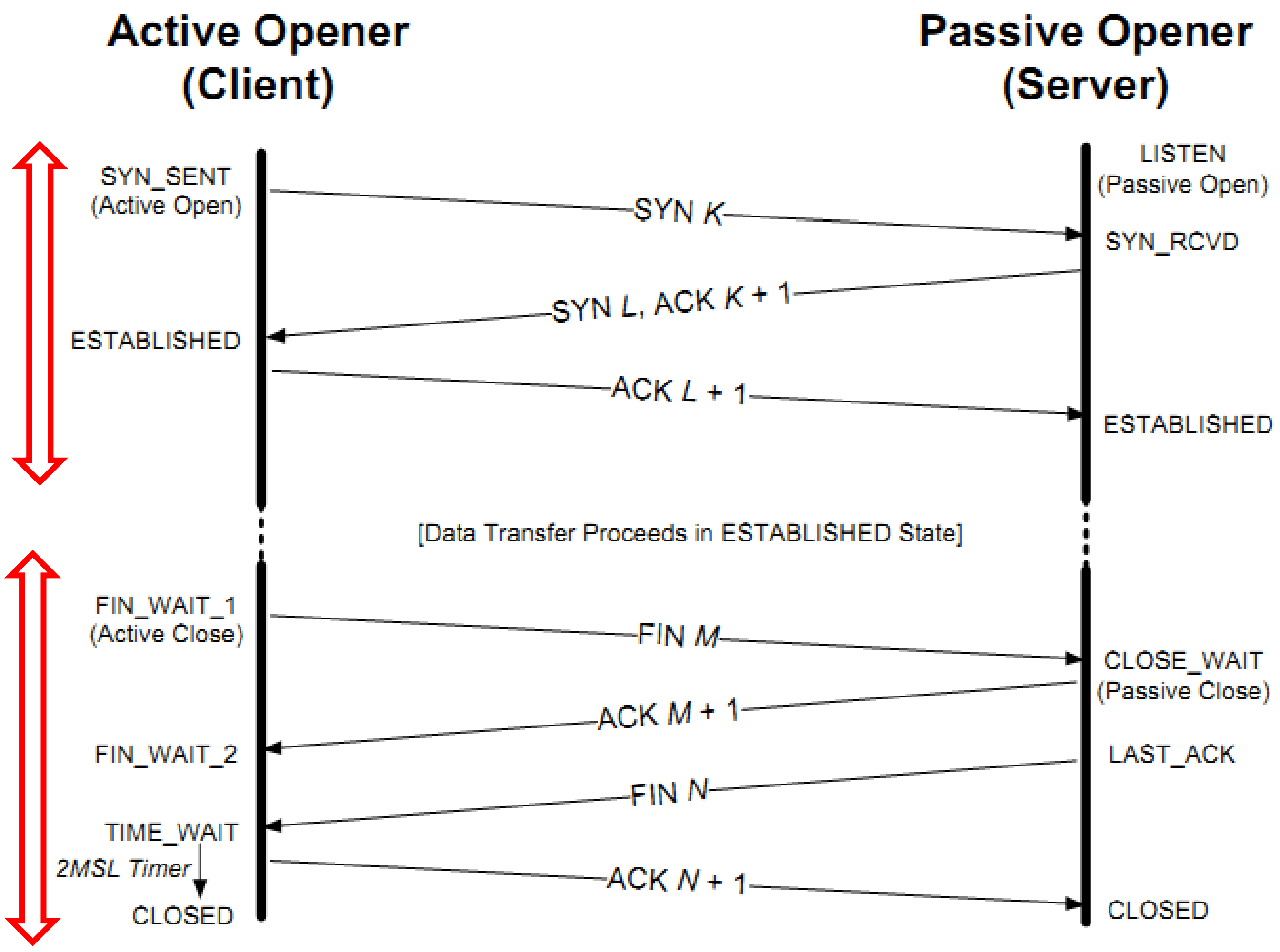
- SYN: A synchronize message, used to initiate and establish a connection. It is so named since one of its functions is to synchronizes sequence numbers between devices

- FIN: A finish message, which is a TCP segment with the FIN bit set, indicating that a device wants to terminate the connection

- ACK: An acknowledgment, indicating receipt of a message such as a SYN or a FIN

http://www.tcpipguide.com/free/t_TCPOperationalOverviewandtheTCPFiniteStateMachineF-2.htm

TCP States

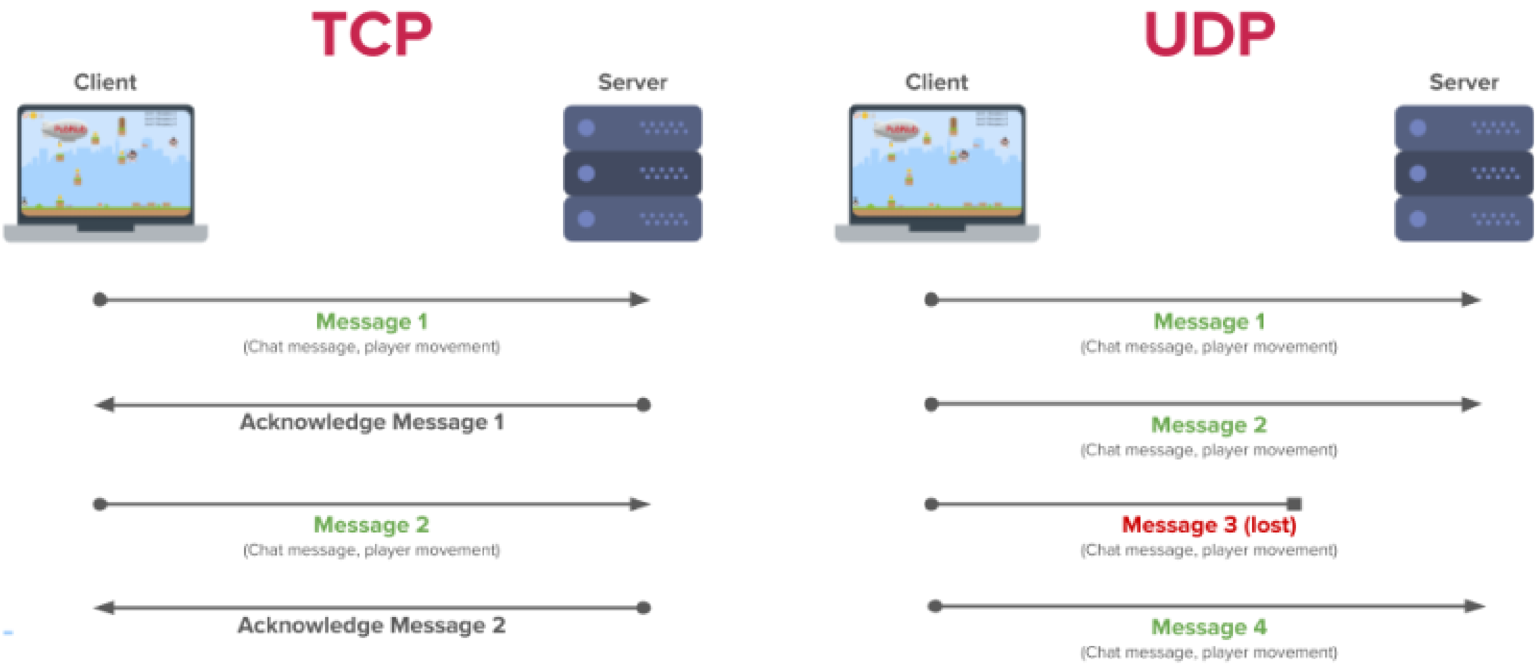


Active Opener
(Client)
Passive Opener
(Server)
SYN_SENT
(Active Open)
LISTEN
(Passive Open)
SYN K
SYN_RCVD
SYN L, ACK K + 1
ESTABLISHED
ACK L + 1
ESTABLISHED
[Data Transfer Proceeds in ESTABLISHED State]
FIN_WAIT_1
(Active Close)
FIN M
CLOSE_WAIT
(Passive Close)
ACK M + 1
FIN_WAIT_2
LAST_ACK
FIN N
TIME_WAIT
2MSL Timer
ACK N + 1
CLOSED
CLOSED

https://www.researchgate.net/publication/320114709_Engineering_problems_in_initial_phase_of_cloud_control_system/figures?lo=1&utm_source=google&utm_medium=organic

TCP and UDP

Protocol	TCP	UDP
Connection	connection-oriented	connectionless
Usage	high reliability, critical-less transmission time	fast, efficient transmission, small queries, huge numbers of clients
Ordering of data packets	rearranges packets in order	no inherent order
Reliability	yes	no
Streaming of data	read as a byte stream	sent and read individually
Error checking	error checking and recovery	simply error checking, no error recovery
Acknowledgement	acknowledgement segments	no acknowledgment



https://ydevern.files.wordpress.com/2018/09/tcp-vs-udp.png?w=809