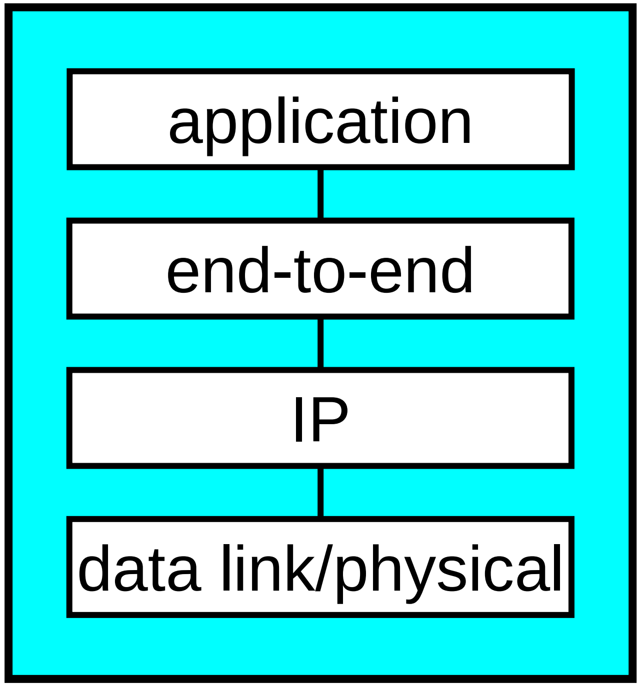
Lecture 10:
End to End Protocols

CS/ECE 438: Communication Networks

Prof. Matthew Caesar

April 2, 2010

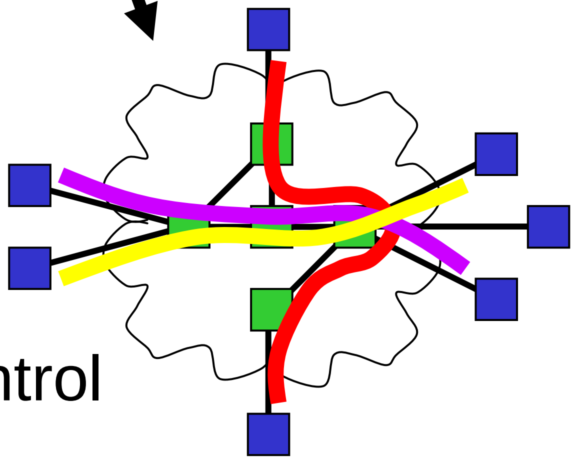
The Big(ger) Picture



428/598 topics

detailed description of issues here

most coverage until now



performance ← congestion control

Where are you?

- Understand how to
  - Build a network on one physical medium
  - Connect networks
  - Implement a reliable byte stream
  - Address network heterogeneity
  - Address global scale
- Final part of class
  - End-to-end issues and common protocols
  - Congestion control: TCP heuristics, switch/router approaches to fairness
  - Performance analysis

End-to-End Protocols

End-to-end Service Model

Protocol Examples

User Datagram Protocol (UDP)

Transmission Control Protocol (TCP)

End-to-End Service Model

- User perspective of network
  - Knowledge of required functionality
  - Implementation is hidden
- Focus
  - Enable communication between applications
  - Translate from host-to-host protocols
- Services
  - Services that cannot be implemented in lower layers (hop-by-hop basis)
  - Avoid duplicate effort
  - Services not needed by all applications

End-to-End Service Model

- Build on "best effort" service provided by network layer (IP)
  - Messages sent from a host are delivered to another host
    - May be lost
    - May be reordered
    - May be delivered multiple times
    - May be limited to a finite size
    - May be delivered after a long delay

End-to-End Service Model

- Support services needed by the application
  - Multiple connections per host
  - Guaranteed delivery
  - Messages delivered in the order they were sent
  - Messages delivered at most once
  - No limit on message size
  - Synchronization between sender and receiver
  - Flow control

End-to-End Service Model

- Challenge
  - Given
    - Less than desirable properties of the underlying network
  - Create
    - High-level services required by applications
- Services
  - Asynchronous demultiplexing service
  - Reliable byte-stream service

User Datagram Protocol (UDP)

- Simple connectionless demultiplexer
  - No handshaking
  - Each segment handled independently
- Service Model
  - Thin veneer over IP services
  - Unreliable unordered datagram service
  - Addresses multiplexing of multiple connections
- Multiplexing
  - 16-bit port numbers
  - Well-known ports
- Checksum
  - Validate header
  - Optional in IPv4
  - Mandatory in IPv6

User Datagram Protocol (UDP)

- Why is there a UDP?
  - No connection establishment
    - Low delay
  - Simple
    - No connection state at sender, receiver
  - Small header
  - No congestion control
    - UDP can blast away as fast as desired
- What kind of applications is UDP good for?
  - Streaming multimedia apps
  - Loss tolerant
  - Rate sensitive
- Other UDP uses
  - DNS, SNMP
- Reliable transfer over UDP
  - At application layer
  - Application-specific error recovery

UDP Header Format

0 8	16 31
Source Port	Destination Port
UDP Length	UDP Checksum

UDP Header Format

0	8	16	31
Source Port		**Destination Port**	
UDP Length		**UDP Checksum**	

- 16-bit source and destination ports

UDP Header Format

0 8	16 31
Source Port	Destination Port
UDP Length	UDP Checksum

- Length includes 8-byte header and data

UDP Header Format

0 – 15	16 – 31
Source Port	Destination Port
UDP Length	UDP Checksum

- Checksum
- Uses IP checksum algorithm
  - Computed on header, data and pseudo header

0 – 7	8 – 15	16 – 31
Source IP Address		
Destination IP Address		
0	17 (UDP)	UDP Length

UDP Header Format

0 8 16 31

Source Port	Destination Port
UDP Length	UDP Checksum

- Checksum
  - What purpose does the checksum serve?
  - Why is it mandatory when using IPv6?

0 8 16 31

Source IP Address		
Destination IP Address		
0	17 (UDP)	UDP Length

Transmission Control Protocol (TCP)

- Reliable byte stream
- Service model
  - Multiple connections per host
  - Guaranteed delivery
  - Messages delivered in the order they were sent
  - Messages delivered at most once
  - No limit on message size
  - Synchronization between sender and receiver
  - Flow control
- Multiplexing
  - Equivalent to UDP
- Checksum
  - Equivalent to UDP
  - Mandatory

TCP

- Connection oriented
  - Explicit setup and teardown required
- Full duplex
  - Data flows in both directions simultaneously
  - Point-to-point connection
- Byte stream abstraction
  - No boundaries in data
  - App writes bytes, TCP send segments, App receives bytes

TCP

- Rate control
  - Flow control to restrict sender rate to something manageable by receiver
  - Congestion control to restrict sender to something manageable by network
  - Both need to handle the presence of other traffic

TCP Outline

- TCP vs. Sliding window on a direct link
- Usage model
- Segment header format and options
- States and state diagram
- Sliding window implementation details
- Flow control issues
- Bit allocation limitations
- Adaptive retransmission algorithms

TCP vs. Direct Link

- Explicit connection setup required
  - Dialup vs. dedicated line
- RTT varies
  - Among peers (host at other end of connection)
  - Over time
  - Requires adaptive approach to retransmission (and window size)
- Packets
  - Delayed
  - Reordered
  - Late

TCP vs. Direct Link

- Peer capabilities vary
  - Minimum link speed on route
  - Buffering capacity at destination
  - Requires adaptive approach to window sizes
- Network capacity varies
  - Other traffic competes for most links
  - Requires global congestion control strategy
- Question
  - Why not implement more functionality (reliability, ordering, congestion control) in IP?

Proposal: Reliable Network Layer

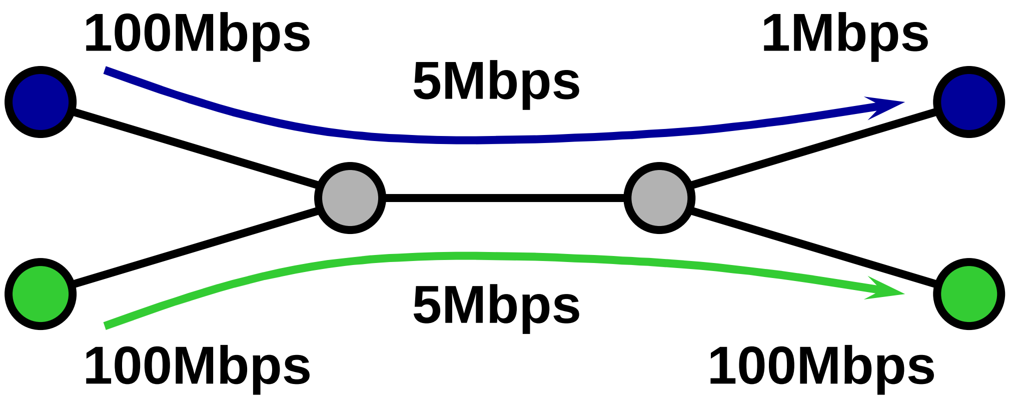
- Service
  - High probabilistic guarantee of correct, in order data transmission at the network layer
  - Hop-by hop network layer ACKs
- Is this sufficient?
- No
  - Routers may crash, buffers may overflow
- Is it beneficial?
  - Maybe, depends on link's error rate
  - Improve performance, not provide correctness

The End-to-End Argument

- Lower layer functions
  - May be redundant or of little value when compared with providing them at that low layer
- Functionality
  - Implemented at a lower layer iff it can be correctly and completely implemented there
- Real constraint
  - Implementing functionality at a lower level should have minimum performance impact on applications that do not use the functionality

End-to-End Argument

- In-order delivery
  - hop-by-hop ordering guarantee is not robust to path changes or multiple paths
- Congestion control
  - Should be stopped at source
  - But network can provide feedback



green should get 9Mbps, but gets only 5Mbps with hop-by-hop drops

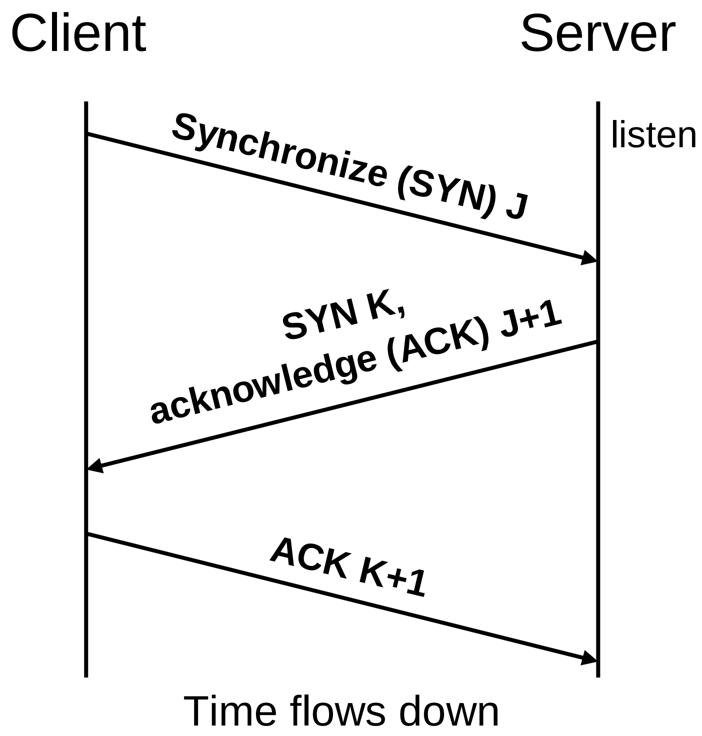
TCP Internals

TCP Usage Model

- Connection setup
  - 3-way handshake
- Data transport
  - Sender writes data
  - TCP
    - Breaks data into segments
    - Sends each segment over IP
    - Retransmits, reorders and removes duplicates as necessary
  - Receiver reads some data
- Teardown
  - 4 step exchange

TCP Connection Establishment

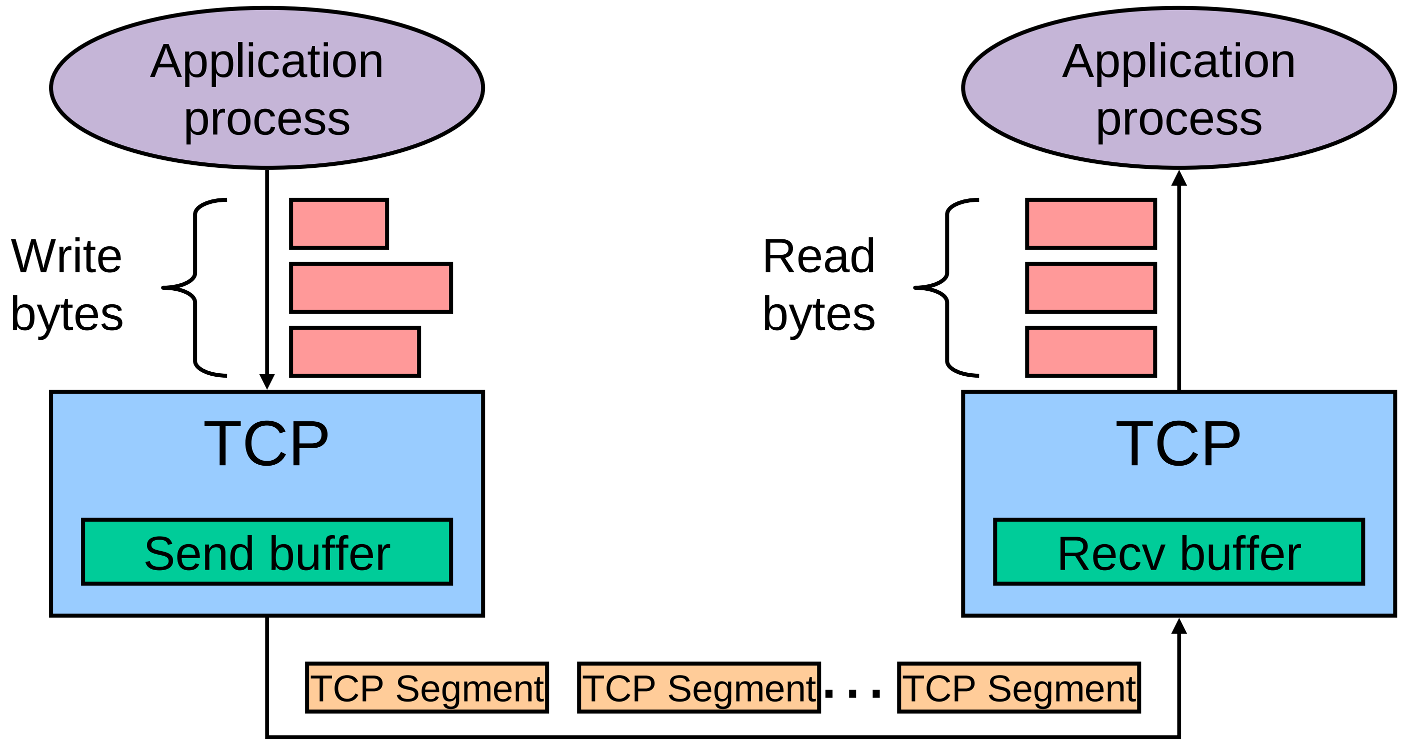
- 3-Way Handshake
  - Sequence Numbers
    - J,K
  - Message Types
    - Synchronize (SYN)
    - Acknowledge (ACK)
  - Passive Open
    - Server listens for connection from client
  - Active Open
    - Client initiates connection to server



TCP Data Transport

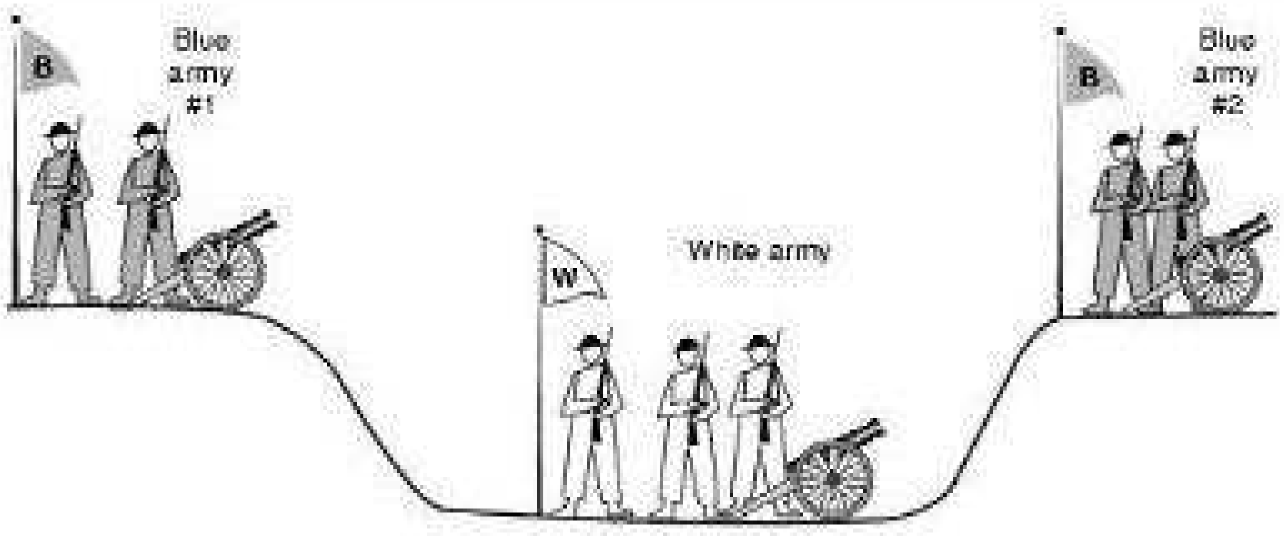
- Data broken into segments
  - Limited by maximum segment size (MSS)
  - Defaults to 352 bytes
  - Negotiable during connection setup
  - Typically set to
    - MTU of directly connected network – size of TCP and IP headers
- Three events cause a segment to be sent
  - ≥ MSS bytes of data ready to be sent
  - Explicit PUSH operation by application
  - Periodic timeout

TCP Byte Stream



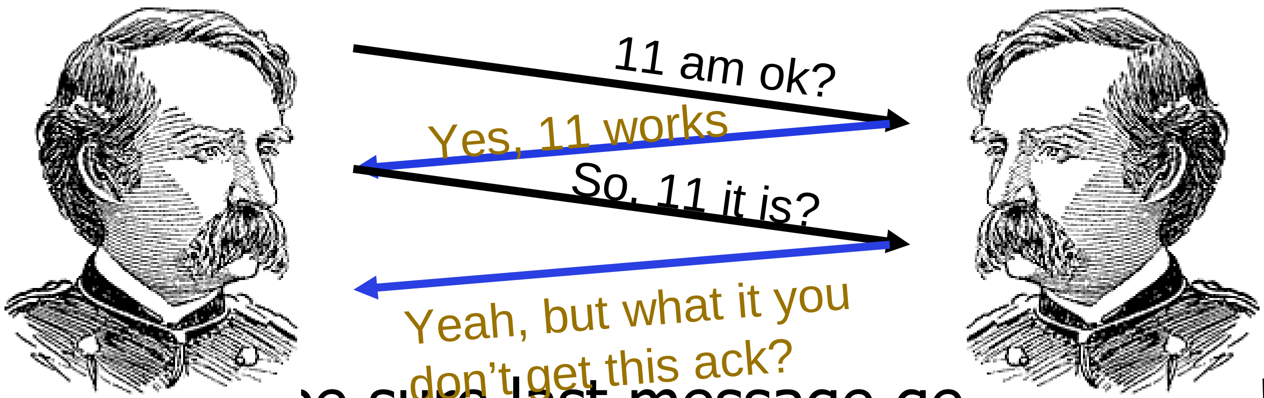
TCP Connection Termination

- Two generals problem
  - Enemy camped in valley
  - Two generals' hills separated by enemy
  - Communication by unreliable messengers
  - Generals need to agree whether to attack or retreat



Two generals problem

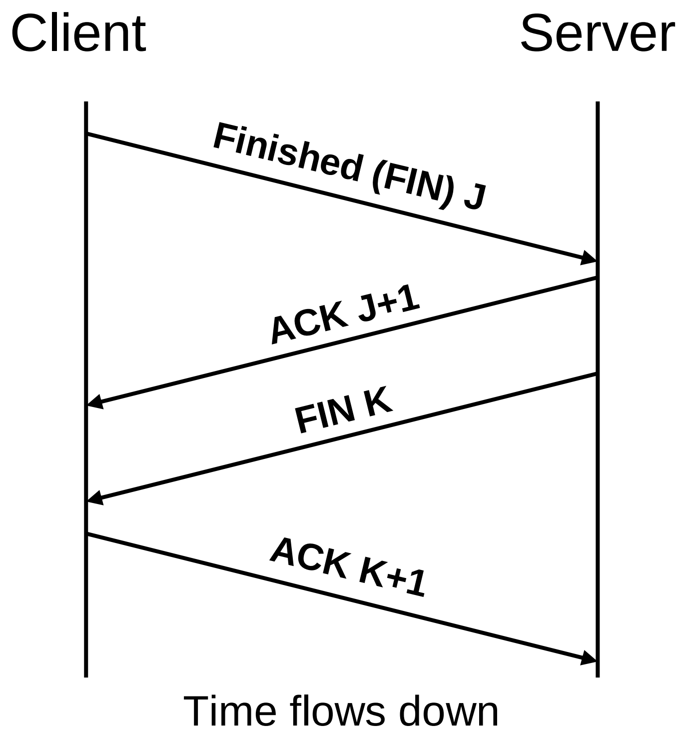
- Can messages over an unreliable network be used to guarantee two entities do something simultaneously?
  - No, even if all messages get through



- No way to be sure last message gets through!

TCP Connection Termination

- Message Types
  - Finished (FIN)
  - Acknowledge (ACK)
- Active Close
  - Sends no more data
- Passive close
  - Accepts no more data



TCP Segment Header Format

<table>
<tr><td>0</td><td>8</td><td>16</td><td>31</td></tr>
<tr><td colspan="3">Source Port</td><td>Destination Port</td></tr>
<tr><td colspan="4">Sequence Number</td></tr>
<tr><td colspan="4">ACK Sequence Number</td></tr>
<tr><td>Header Length</td><td>0</td><td>Flags</td><td>Advertised Window</td></tr>
<tr><td colspan="3">TCP Checksum</td><td>Urgent Pointer</td></tr>
<tr><td colspan="4">Options</td></tr>
</table>

TCP Segment Header Format

0	8		16	31
Source Port			**Destination Port**	
Sequence Number				
ACK Sequence Number				
Header Length	0	Flags	Advertised Window	
TCP Checksum			Urgent Pointer	
Options				

- 16-bit source and destination ports

TCP Segment Header Format

<table>
<tr><th>0</th><th>8</th><th>16</th><th>31</th></tr>
<tr><td colspan="2">Source Port</td><td colspan="2">Destination Port</td></tr>
<tr><td colspan="4">Sequence Number</td></tr>
<tr><td colspan="4">ACK Sequence Number</td></tr>
<tr><td>Header Length</td><td>0 | Flags</td><td colspan="2">Advertised Window</td></tr>
<tr><td colspan="2">TCP Checksum</td><td colspan="2">Urgent Pointer</td></tr>
<tr><td colspan="4">Options</td></tr>
</table>

- 32-bit send and ACK sequence numbers

TCP Segment Header Format

0	8		16 … 31
Source Port			Destination Port
Sequence Number			
ACK Sequence Number			
Header Length	0	Flags	Advertised Window
TCP Checksum			Urgent Pointer
Options			

- 4-bit header length in 4-byte words
  - Minimum 5 bytes
  - Offset to first data byte

TCP Segment Header Format

0	8		16 31
Source Port			Destination Port
Sequence Number			
ACK Sequence Number			
Header Length	0	Flags	Advertised Window
TCP Checksum			Urgent Pointer
Options			

- Reserved
  - Must be 0

TCP Segment Header Format

<table>
<tr><td>0</td><td>8</td><td>16</td><td>31</td></tr>
<tr><td colspan="3">Source Port</td><td>Destination Port</td></tr>
<tr><td colspan="4">Sequence Number</td></tr>
<tr><td colspan="4">ACK Sequence Number</td></tr>
<tr><td>Header Length</td><td>0</td><td>Flags</td><td>Advertised Window</td></tr>
<tr><td colspan="3">TCP Checksum</td><td>Urgent Pointer</td></tr>
<tr><td colspan="4">Options</td></tr>
</table>

- 6 1-bit flags

URG: Contains urgent data
ACK: Valid ACK seq. number
PSH: Do not delay data delivery

RST: Reset connection
SYN: Synchronize for setup
FIN: Final segment for teardown

TCP Segment Header Format

<table>
<tr><td>0</td><td>8</td><td>16</td><td>31</td></tr>
<tr><td colspan="2">Source Port</td><td colspan="2">Destination Port</td></tr>
<tr><td colspan="4">Sequence Number</td></tr>
<tr><td colspan="4">ACK Sequence Number</td></tr>
<tr><td>Header Length</td><td>0</td><td>Flags</td><td>Advertised Window</td></tr>
<tr><td colspan="2">TCP Checksum</td><td colspan="2">Urgent Pointer</td></tr>
<tr><td colspan="4">Options</td></tr>
</table>

- 16-bit advertised window
  - Space remaining in receive window

TCP Segment Header Format

0	8		16	31
Source Port			Destination Port	
Sequence Number				
ACK Sequence Number				
Header Length	0	Flags	Advertised Window	
TCP Checksum			Urgent Pointer	
Options				

- 16-bit checksum
  - Uses IP checksum algorithm
  - Computed on header, data and pseudo header

0	8	16	31
Source IP Address			
Destination IP Address			
0	**16 (TDP)**	**TCP Segment Length**	

TCP Segment Header Format

0	8		16	31
Source Port			Destination Port	
Sequence Number				
ACK Sequence Number				
Header Length	0	Flags	Advertised Window	
TCP Checksum			**Urgent Pointer**	
Options				

- 16-bit urgent data pointer
  - If URG = 1
  - Index of last byte of urgent data in segment

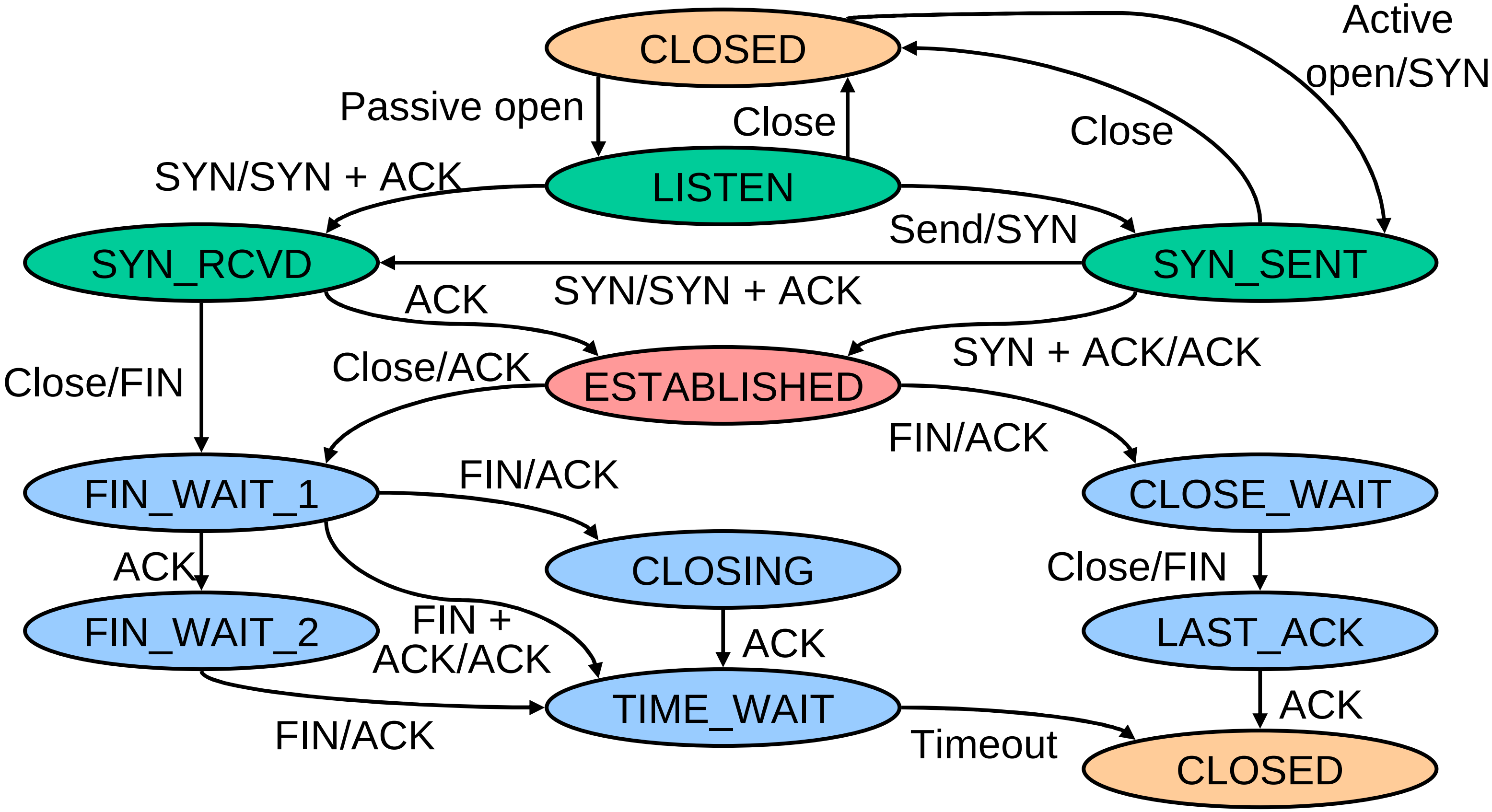
TCP Options

- Negotiate maximum segment size (MSS)
  - Each host suggests a value
  - Minimum of two values is chosen
  - Prevents IP fragmentation over first and last hops
- Packet timestamp
  - Allows RTT calculation for retransmitted packets
  - Extends sequence number space for identification of stray packets
- Negotiate advertised window granularity
  - Allows larger windows
  - Good for routes with large bandwidth-delay products

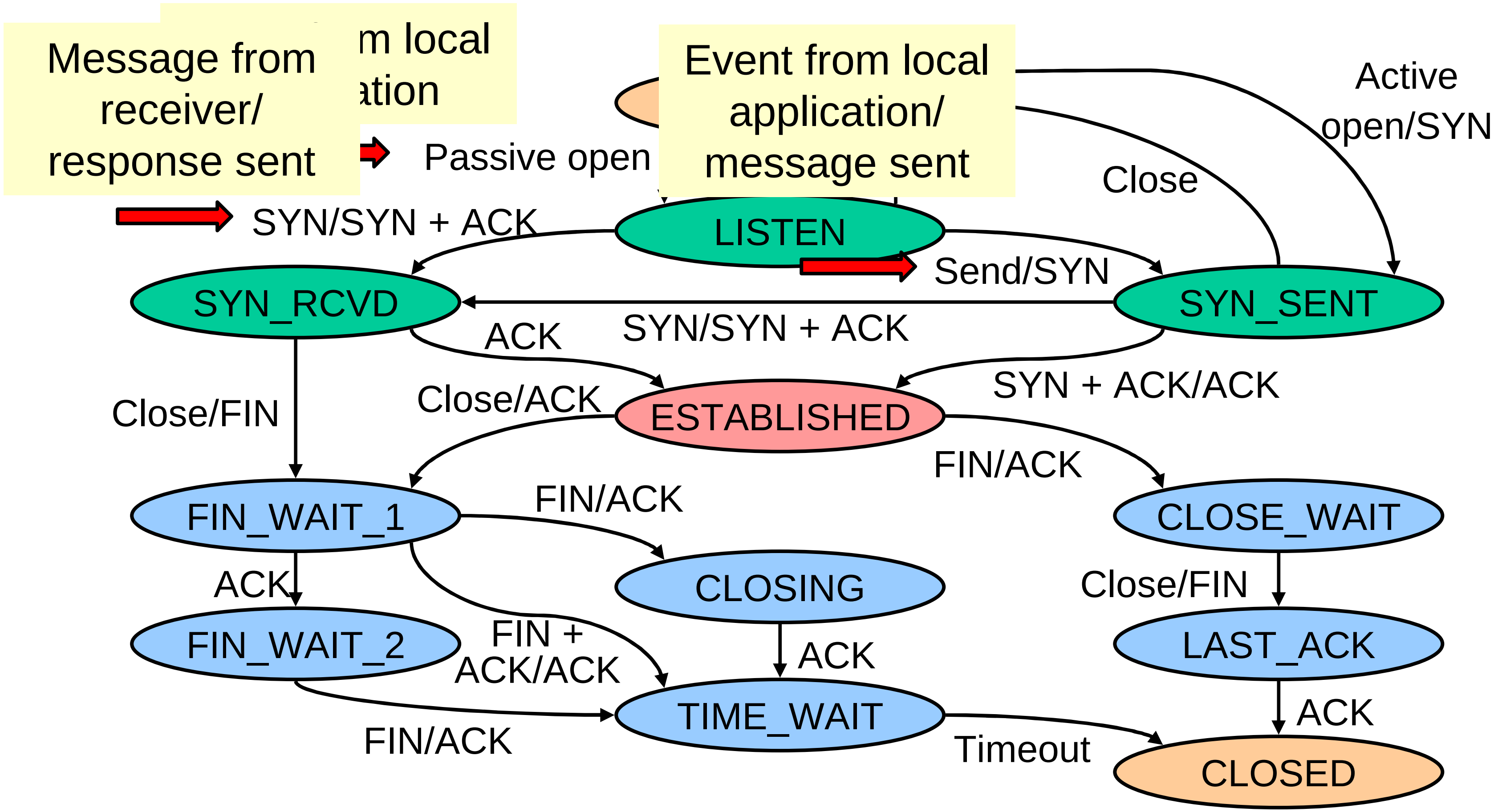
TCP State Descriptions

CLOSED	Disconnected
LISTEN	Waiting for incoming connection
SYN_RCVD	Connection request received
SYN_SENT	Connection request sent
ESTABLISHED	Connection ready for data transport
CLOSE_WAIT	Connection closed by peer
LAST_ACK	Connection closed by peer, closed locally, await ACK
FIN_WAIT_1	Connection closed locally
FIN_WAIT_2	Connection closed locally and ACK'd
CLOSING	Connection closed by both sides simultaneously
TIME_WAIT	Wait for network to discard related packets

TCP State Transition Diagram



TCP State Transition Diagram

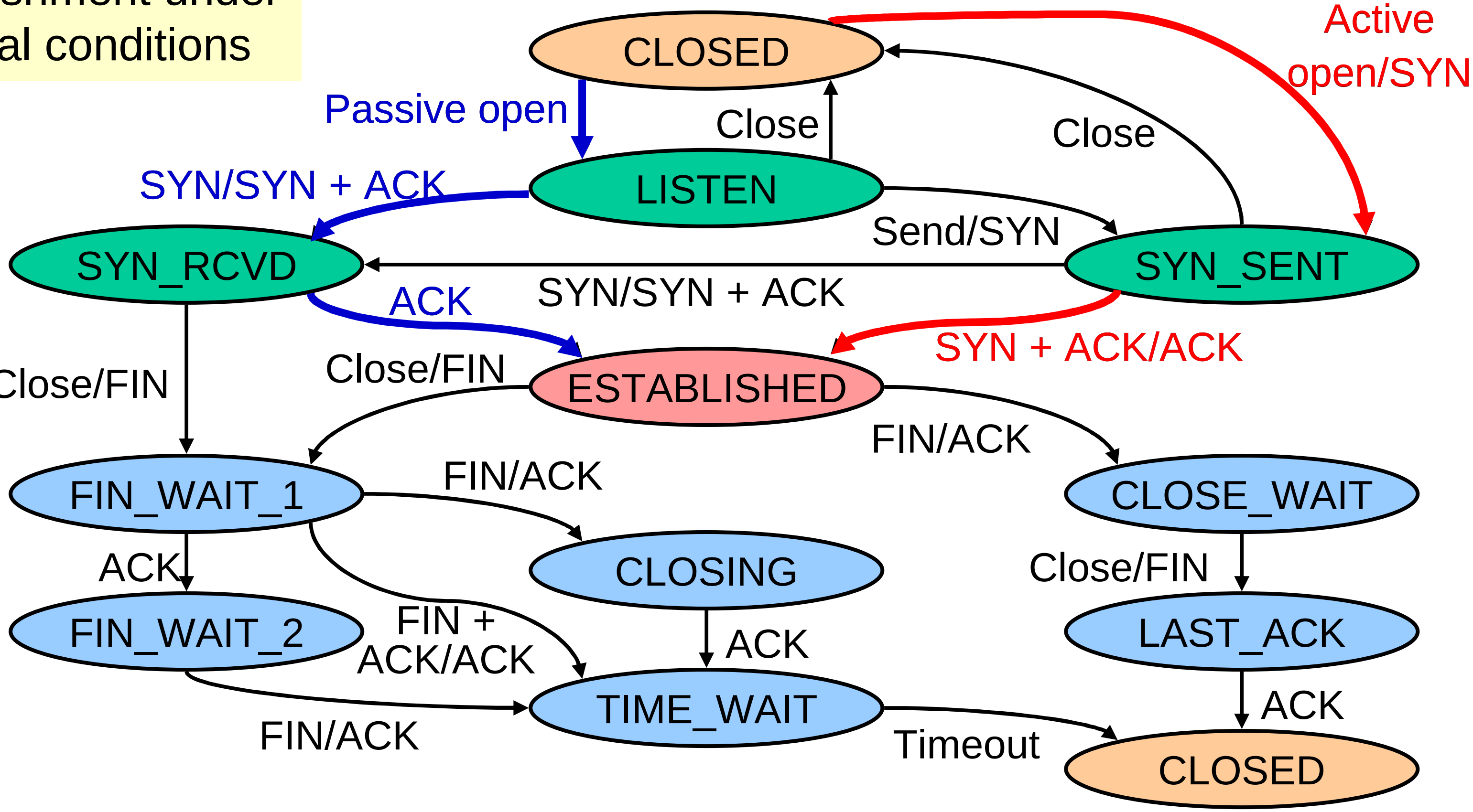


TCP State Transition Diagram

- Questions
  - State transitions
    - Describe the path taken by a server under normal conditions
    - Describe the path taken by a client under normal conditions
    - Describe the path taken assuming the client closes the connection first
  - TIME_WAIT state
    - What purpose does this state serve
    - Prove that at least one side of a connection enters this state
    - Explain how both sides might enter this state

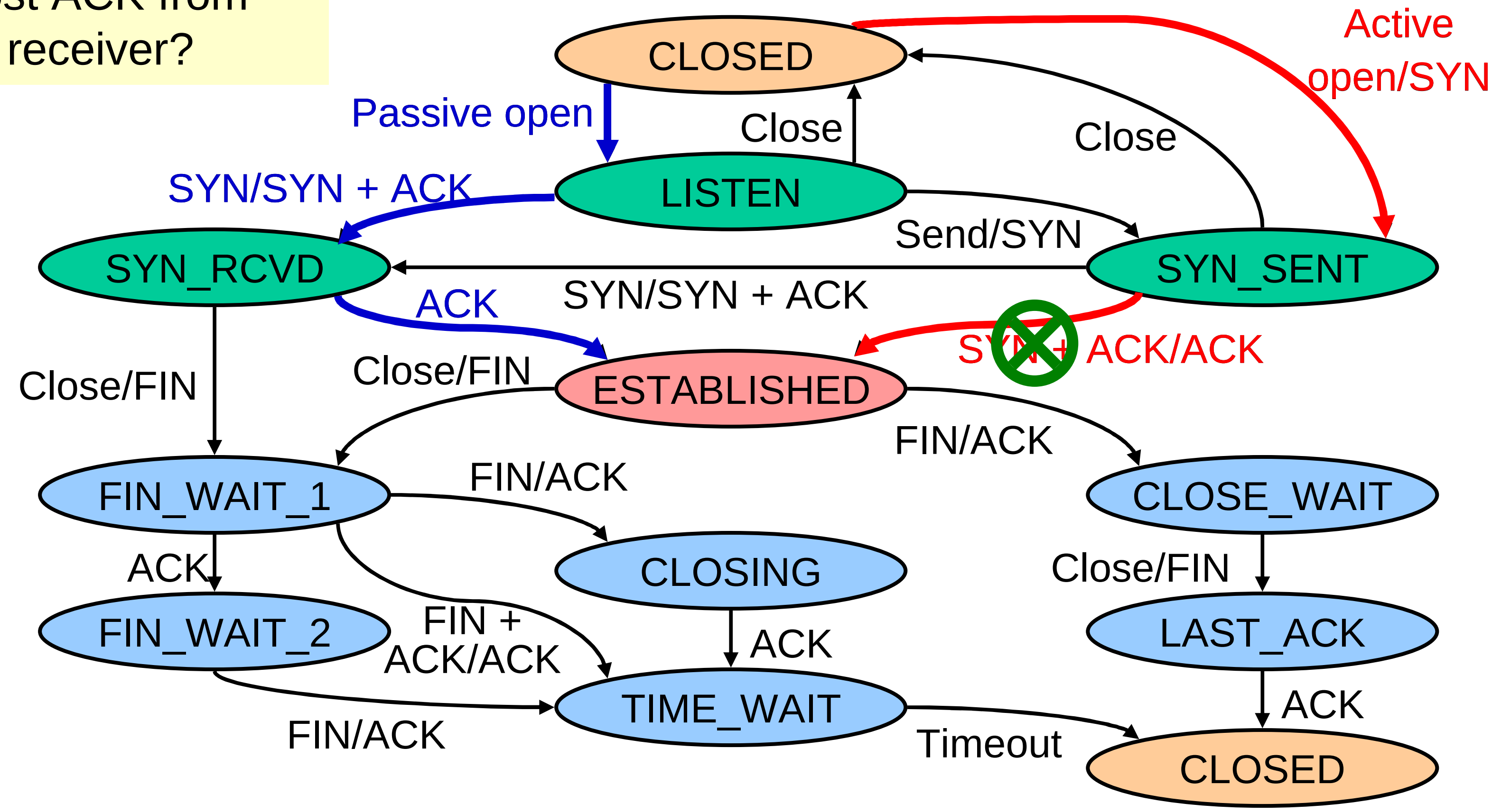
TCP State Transition Diagram

Establishment under normal conditions

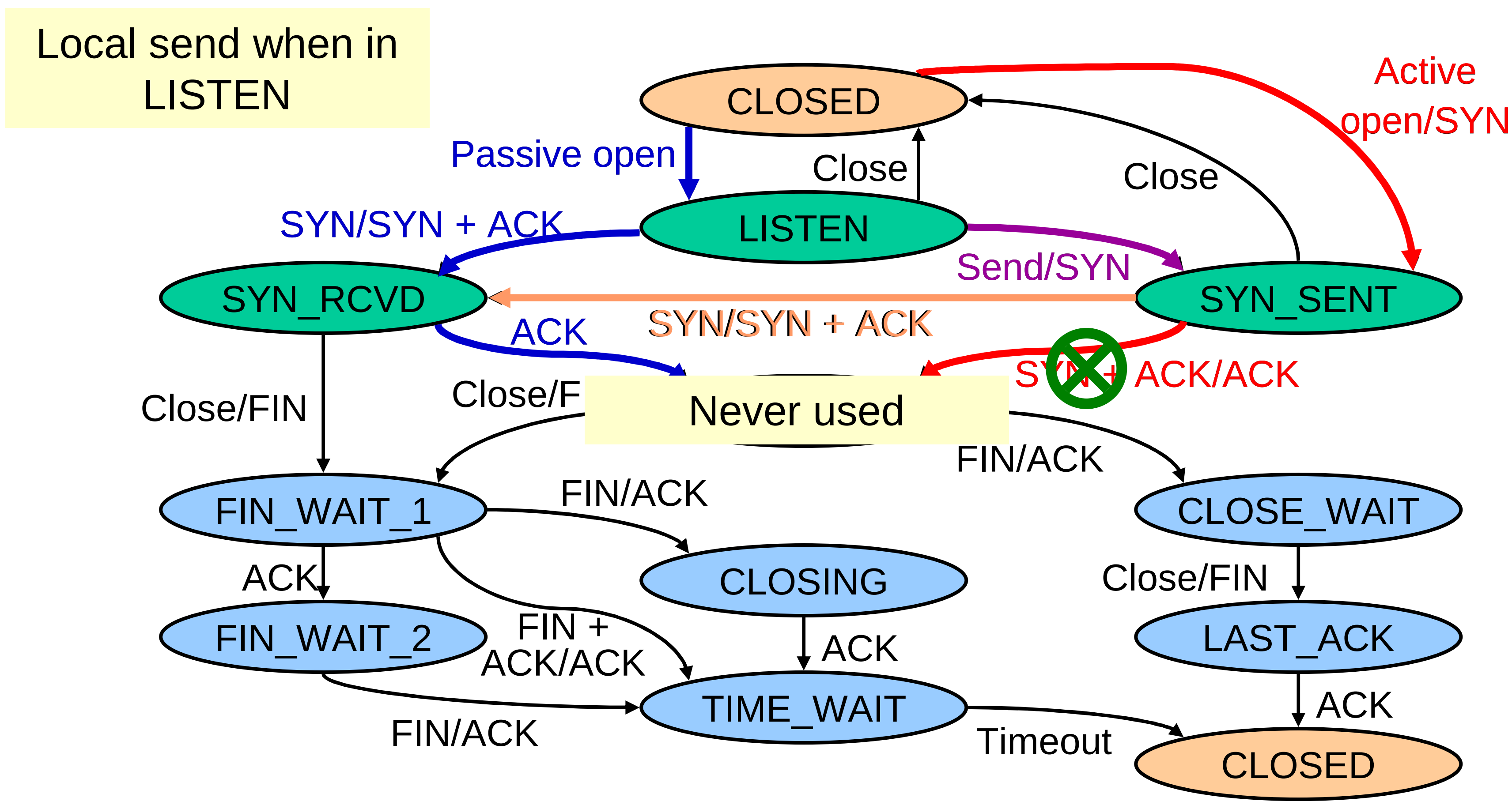


TCP State Transition Diagram

Lost ACK from receiver?

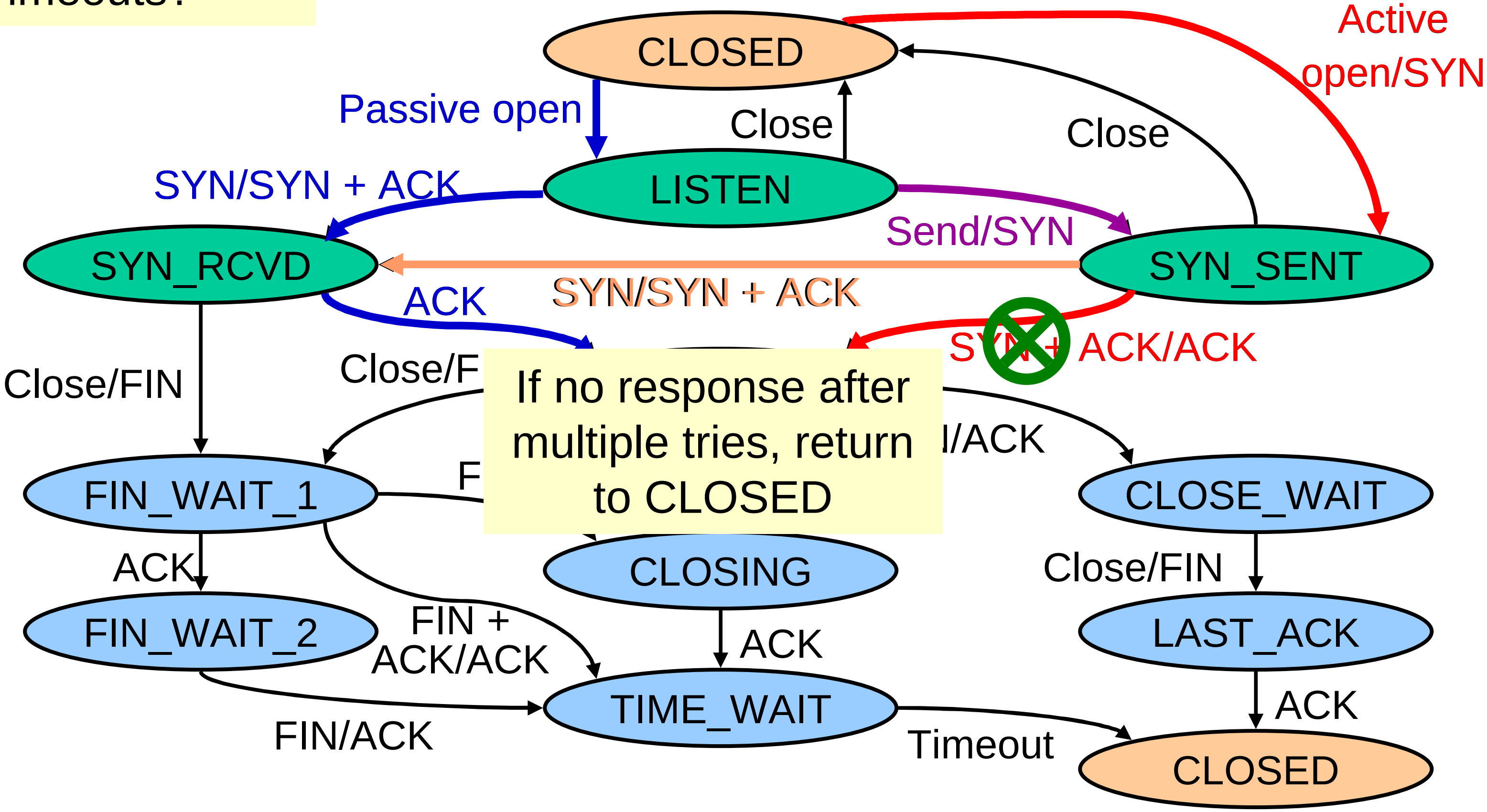


TCP State Transition Diagram



TCP State Transition Diagram

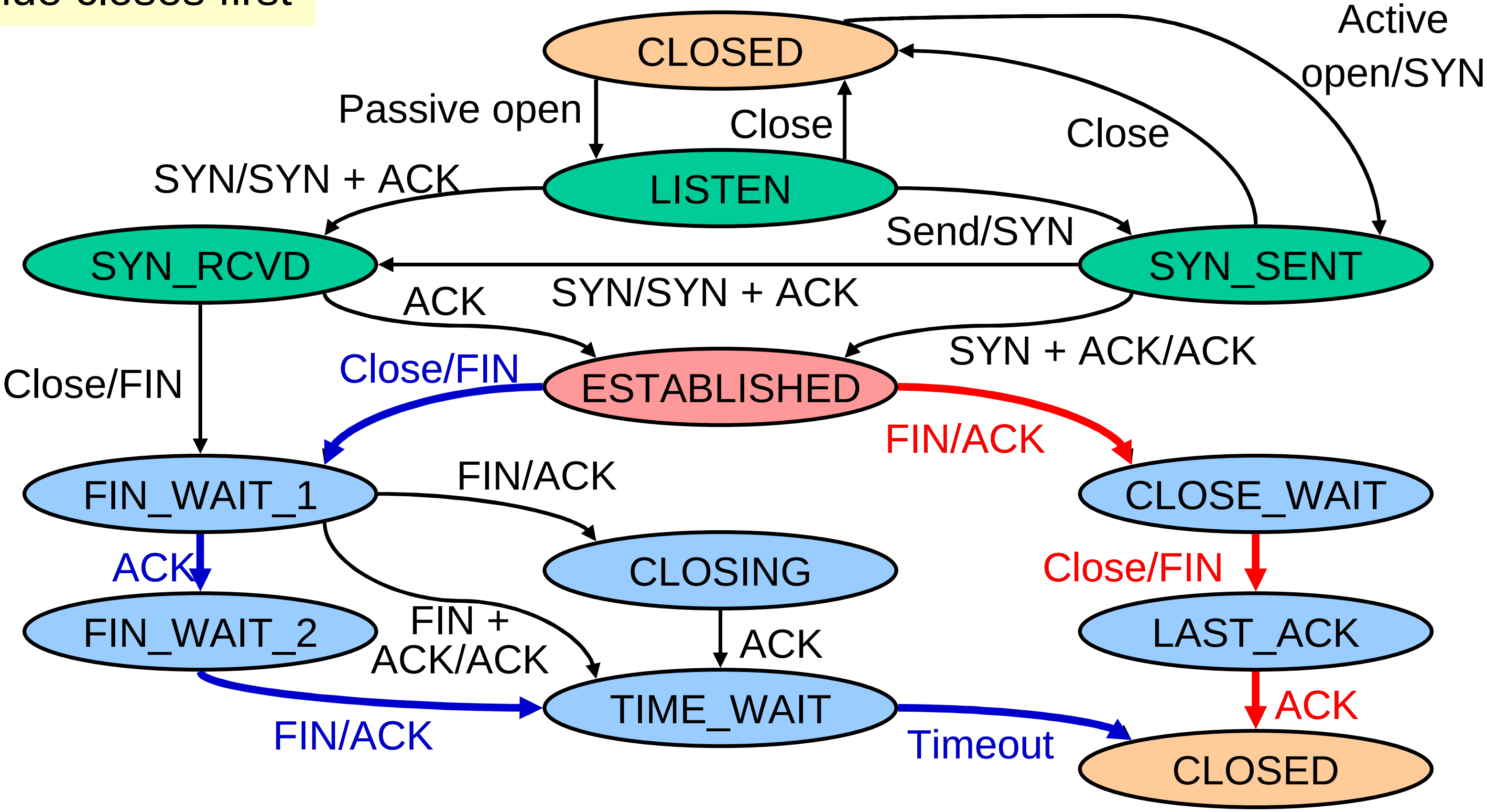
Timeouts?



If no response after multiple tries, return to CLOSED

TCP State Transition Diagram

One side closes first

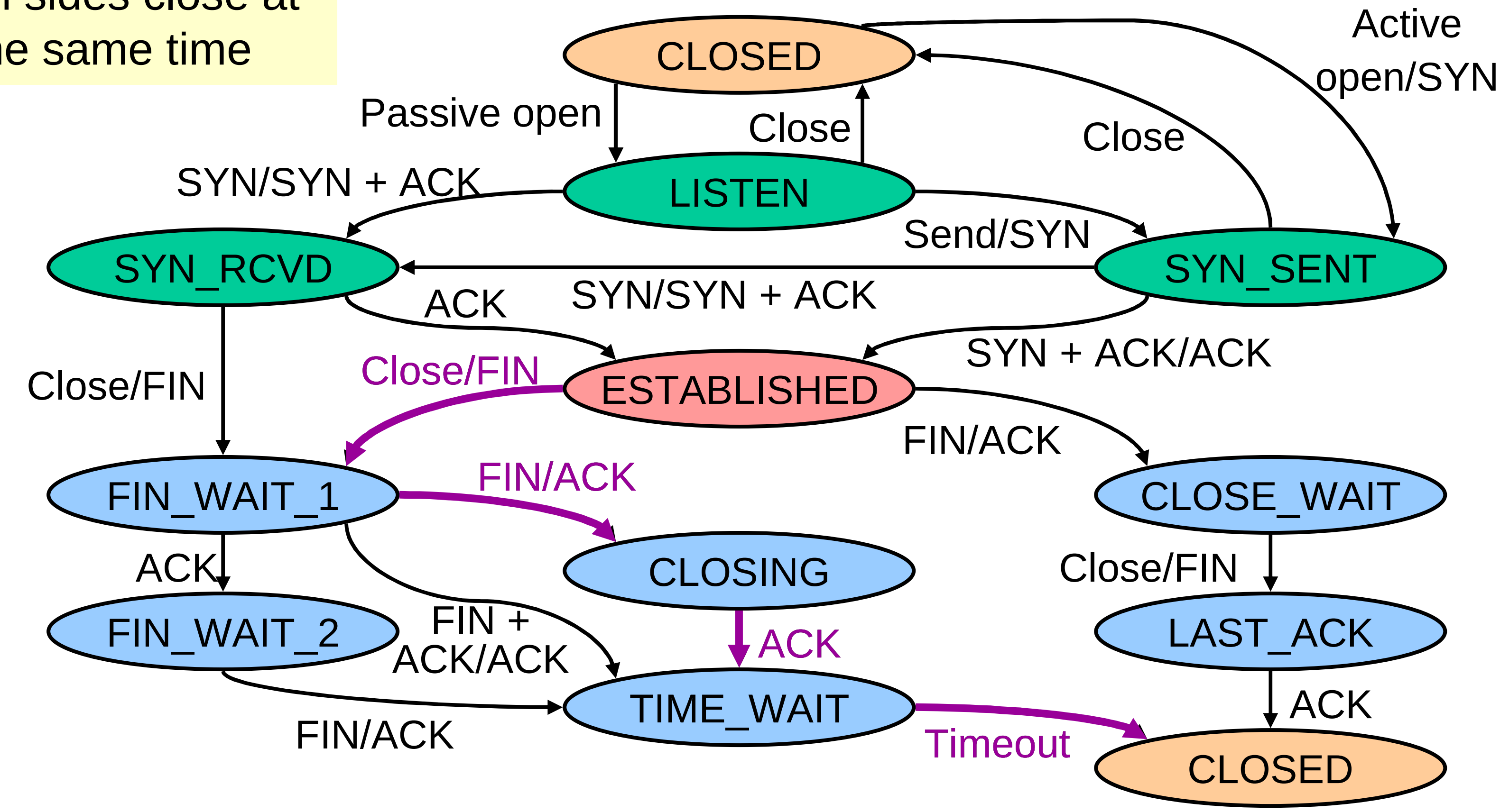


TCP TIME_WAIT State

- Problem
  - What happens if a segment from an old connection arrives at a new connection?
- Maximum Segment Lifetime
  - Max time an old segment can live in the Internet
- TIME_WAIT State
  - Connection remains in this state from two times the maximum segment lifetime

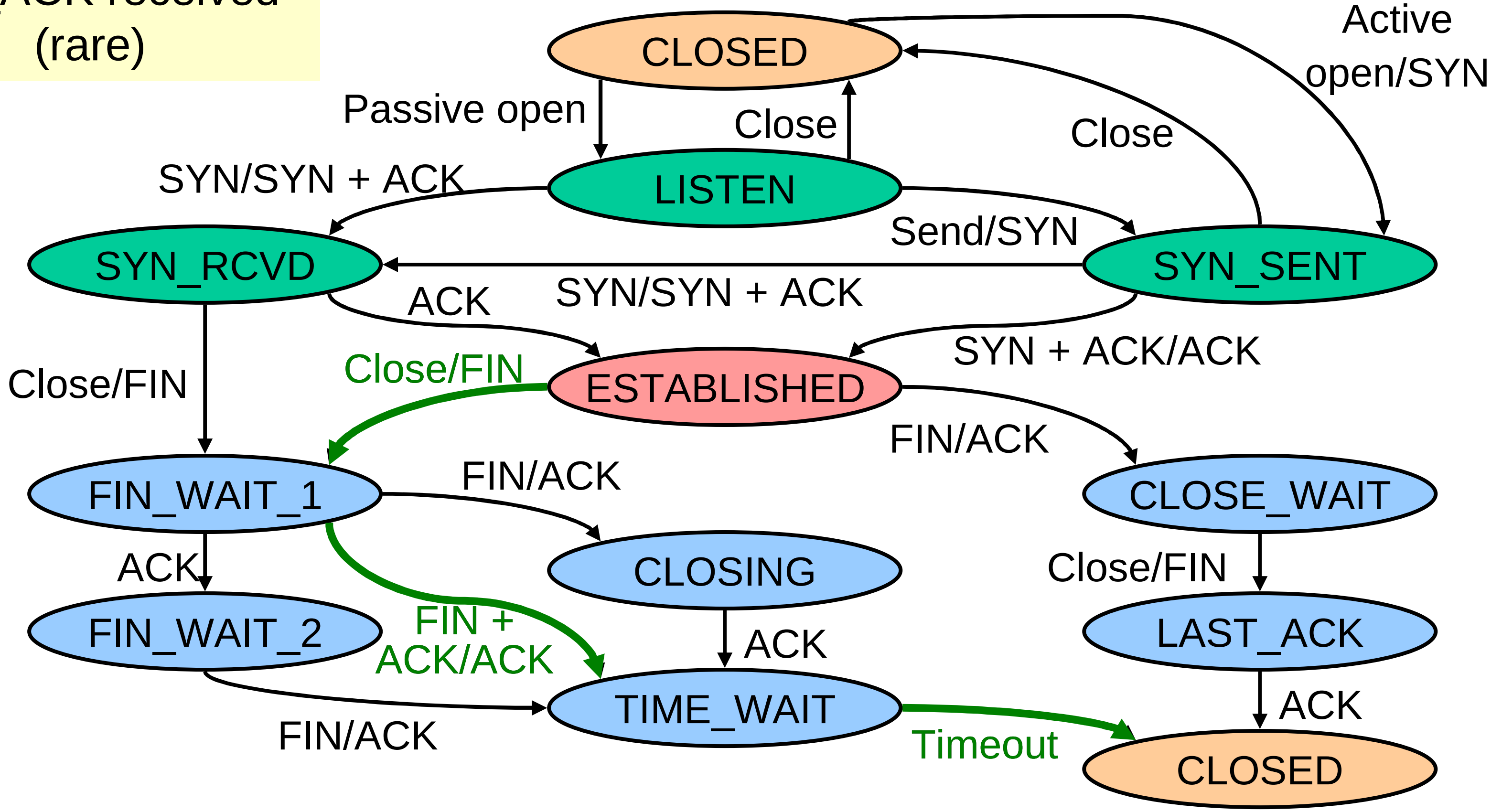
TCP State Transition Diagram

Both sides close at the same time



TCP State Transition Diagram

FIN_ACK received
(rare)



TCP Sliding Window Protocol

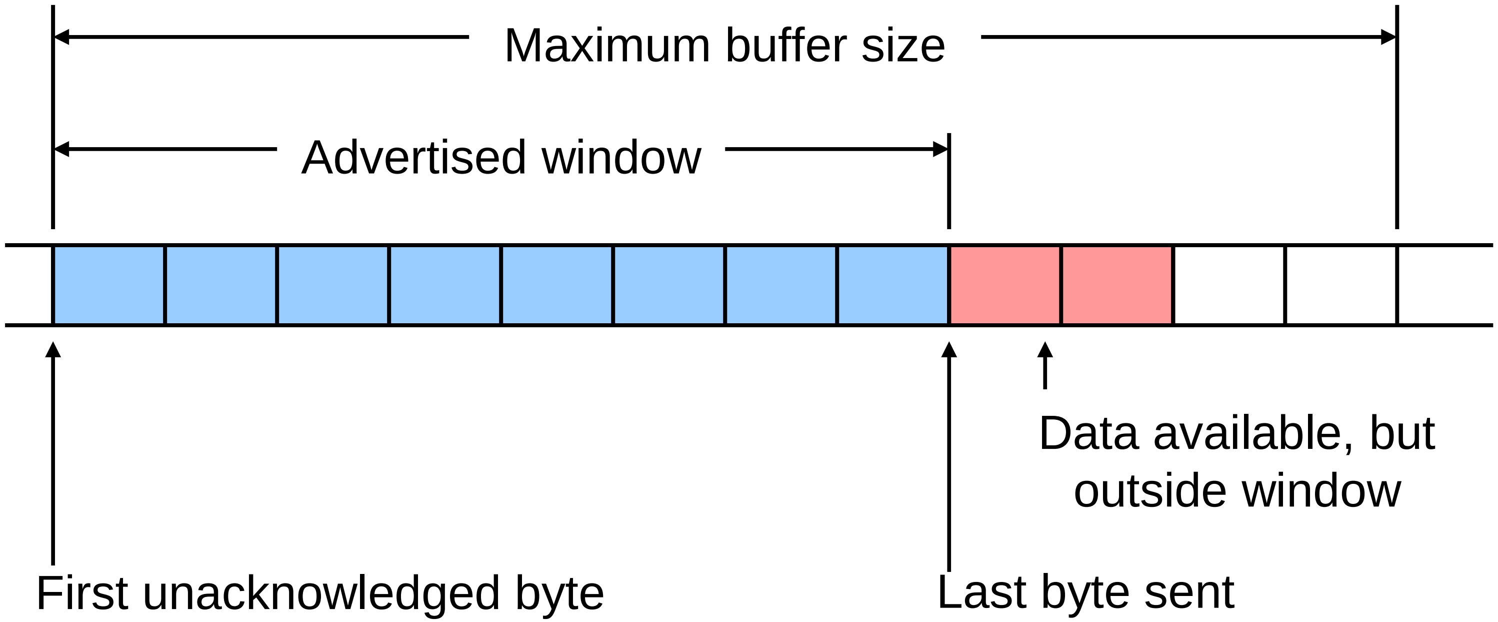
- Sequence numbers
  – Indices into byte stream
- Initial Sequence Number
  – Why not just use 0?
- ACK sequence number
  – Actually next byte expected as opposed to last byte received

TCP Sliding Window Protocol

- Advertised window
  - Enables dynamic receive window size
- Receive buffers
  - Data ready for delivery to application until requested
  - Out-of-order data to maximum buffer capacity
- Sender buffers
  - Unacknowledged data
  - Unsent data out to maximum buffer capacity

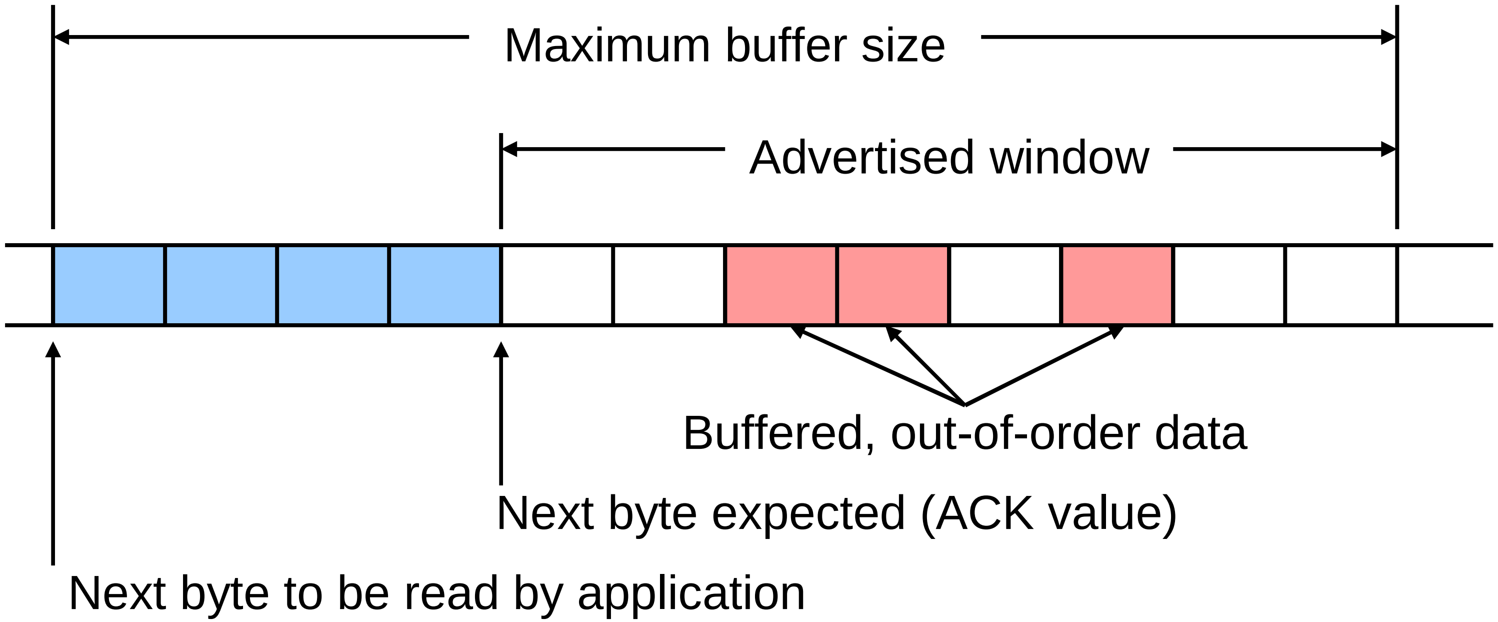
TCP Sliding Window Protocol – Sender Side

- `LastByteAcked <= LastByteSent`
- `LastByteSent <= LastByteWritten`
- Buffer bytes between `LastByteAcked` and `LastByteWritten`



TCP Sliding Window Protocol – Receiver Side

- `LastByteRead <  NextByteExpected`
- `NextByteExpected <= LastByteRcvd + 1`
- Buffer bytes between **`NextByteRead`** and **`LastByteRcvd`**



Flow Control vs. Congestion Control

- Flow control
  - Preventing senders from overrunning the capacity of the receivers
- Congestion control
  - Preventing too much data from being injected into the network, causing switches or links to become overloaded
- Which one does TCP provide?
- TCP provides both
  - Flow control based on advertised window
  - Congestion control discussed later in class

TCP Flow Control: Receiver

- Receive buffer size
  - = **`MaxRcvBuffer`**
  - **`LastByteRcvd - LastByteRead < = MaxRcvBuf`**
- Advertised window
  - **`= MaxRcvBuf - (NextByteExp - NextByteRead)`**
  - Shrinks as data arrives and
  - Grows as the application consumes data

TCP Flow Control: Sender

- Send buffer size
  - = **MaxSendBuffer**
  - **LastByteSent - LastByteAcked < = AdvertWindow**
- Effective buffer
  - **= AdvertWindow - (LastByteSent - LastByteAck)**
  - **EffectiveWindow > 0 to send data**

- Relationship between sender and receiver
  - **LastByteWritten - LastByteAcked < = MaxSendBuffer**
  - **block sender if (LastByteWritten - LastByteAcked) + y > MaxSenderBuffer**

TCP Flow Control

- Problem: Slow receiver application
  - Advertised window goes to 0
  - Sender cannot send more data
  - Non-data packets used to update window
  - Receiver may not spontaneously generate update or update may be lost
- Solution
  - Sender periodically sends 1-byte segment, ignoring advertised window of 0
  - Eventually window opens
  - Sender learns of opening from next ACK of 1-byte segment

TCP Flow Control

- Problem: Application delivers tiny pieces of data to TCP
  - Example: telnet in character mode
  - Each piece sent as a segment, returned as ACK
  - Very inefficient
- Solution
  - Delay transmission to accumulate more data
  - Nagle's algorithm
    - Send first piece of data
    - Accumulate data until first piece ACK'd
    - Send accumulated data and restart accumulation
    - Not ideal for some traffic (e.g., mouse motion)

TCP Flow Control

- Problem: Slow application reads data in tiny pieces
  - Receiver advertises tiny window
  - Sender fills tiny window
  - Known as silly window syndrome
- Solution
  - Advertise window opening only when MSS or ½ of buffer is available
  - Sender delays sending until window is MSS or ½ of receiver's buffer (estimated)

TCP Bit Allocation Limitations

- Sequence numbers vs. packet lifetime
  - Assumed that IP packets live less than 60 seconds
  - Can we send 2^{32} bytes in 60 seconds?
  - Less than an STS-12 line
- Advertised window vs. delay-bandwidth
  - Only 16 bits for advertised window
  - Cross-country RTT = 100 ms
  - Adequate for only 5.24 Mbps!

TCP Sequence Numbers – 32-bit

Bandwidth	Speed	Time until wrap around
T1	1.5 Mbps	6.4 hours
Ethernet	10 Mbps	57 minutes
T3	45 Mbps	13 minutes
FDDI	100 Mbps	6 minutes
STS-3	155 Mbps	4 minutes
STS-12	622 Mbps	55 seconds
STS-24	1.2 Gbps	28 seconds

TCP Advertised Window – 16-bit

Bandwidth	Speed	Delay x Bandwidth Product
T1	1.5 Mbps	18 KB
Ethernet	10 Mbps	122 KB
T3	45 Mbps	549 KB
FDDI	100 Mbps	1.2 MB
STS-3	155 Mbps	1.8 MB
STS-12	622 Mbps	7.4 MB
STS-24	1.2 Gbps	14.8 MB

TCP Round Trip Time and Timeout

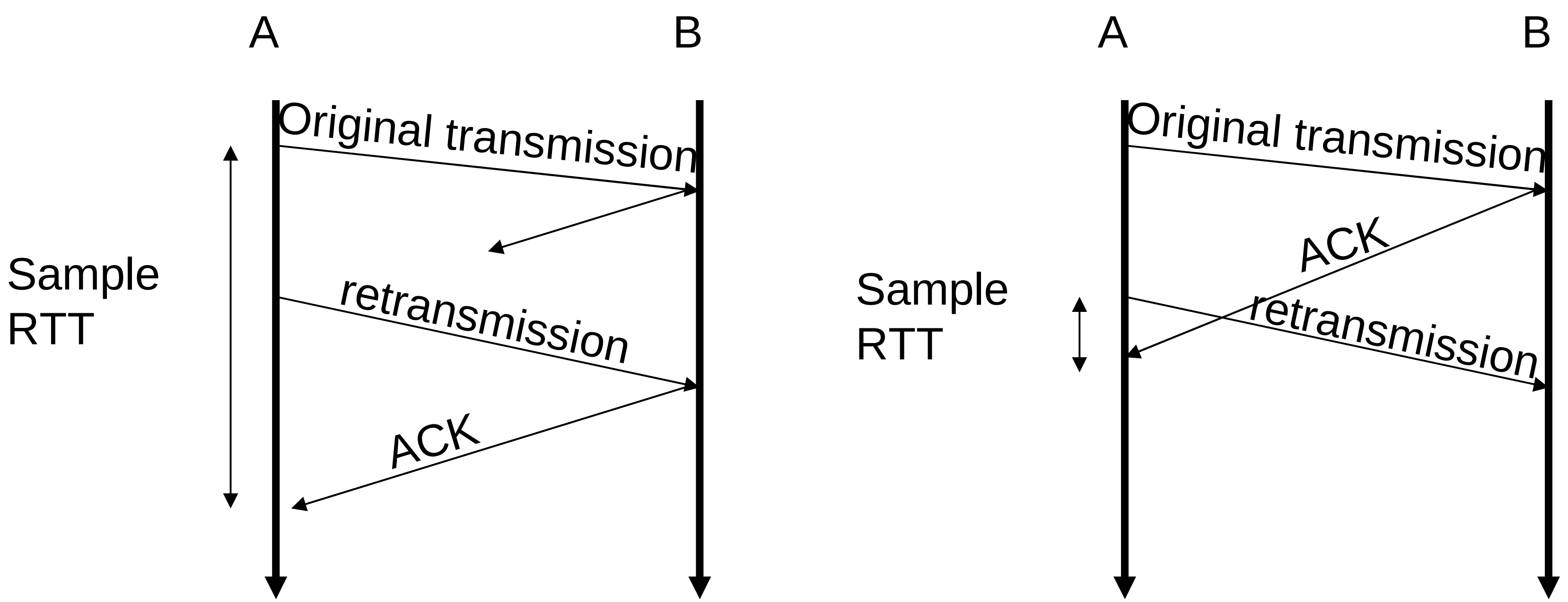
- How should TCP set its timeout value?
  - Longer than RTT
    - But RTT varies
  - Too short
    - Premature timeout
    - Unnecessary retransmissions
  - Too long
    - Slow reaction to segment loss
- Estimating RTT
  - SampleRTT
    - Measured time from segment transmission until ACK receipt
    - Will vary
    - Want smoother estimated RTT
  - Average several recent measurements
    - Not just current SampleRTT

TCP Adaptive Retransmission Algorithm - Original

- Theory
  - Estimate RTT
  - Multiply by 2 to allow for variations
- Practice
  - Use exponential moving average (α = 0.1 to 0.2)
  - Estimate = (α) * measurement + (1- α) * estimate
  - Influence of past sample decreases exponentially fast

TCP Adaptive Retransmission Algorithm - Original

- Problem: What does an ACK really ACK?
  - Was ACK in response to first, second, etc transmission?



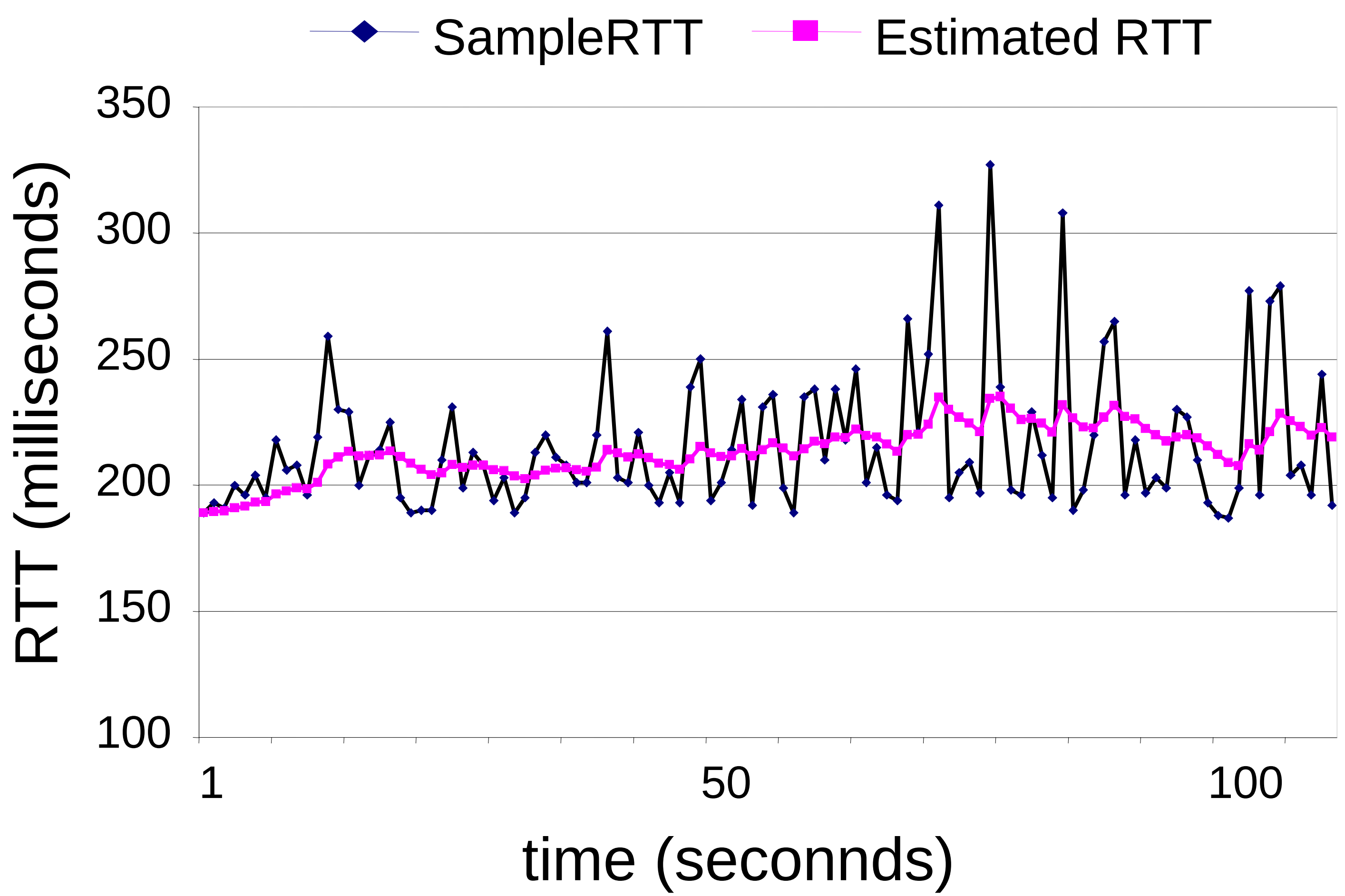
TCP Adaptive Retransmission Algorithm – Karn-Partridge

- Algorithm
  - Exclude retransmitted packets from RTT estimate
  - For each retransmission
    - Double RTT estimate
    - Exponential backoff from congestion

TCP Adaptive Retransmission Algorithm – Karn-Partridge

- Problem
  - Still did not handle variations well
  - Did not solve network congestion problems as well as desired
    - At high loads round trip variance is high

Example RTT Estimation



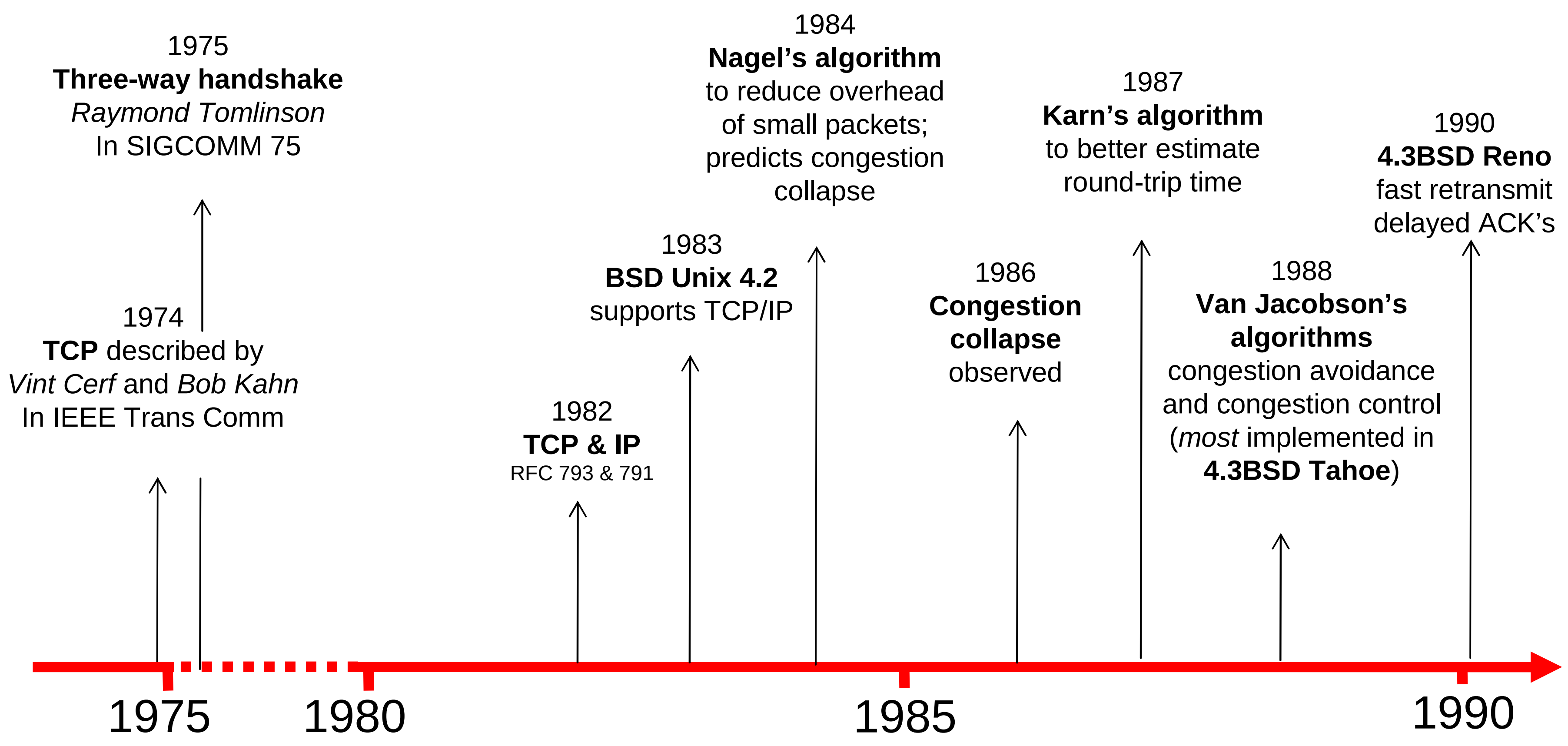
TCP Adaptive Retransmission Algorithm – Jacobson

- Algorithm
  - Estimate variance of RTT
    - Calculate mean interpacket RTT deviation to approximate variance
    - Use second exponential moving average
    - Dev = (β) * |RTT_Est – Sample| + (1–β) * Dev
    - β = 0.25, A = 0.125 for RTT_est
  - Use variance estimate as component of RTT estimate
    - Next_RTT = RTT_Est + 4 * Dev
  - Protects against high jitter

TCP Adaptive Retransmission Algorithm – Jacobson

- Notes
  - Algorithm is only as good as the granularity of the clock
  - Accurate timeout mechanism is important for congestion control

Evolution of TCP



TCP Through the 1990s