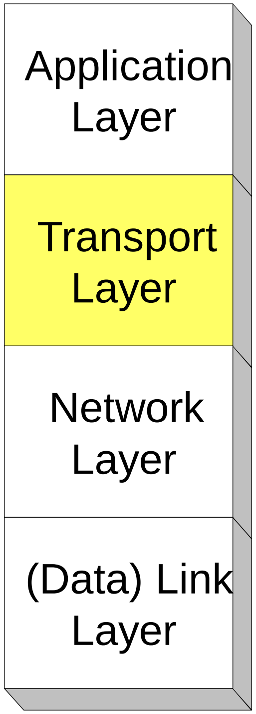
ECE 595
Computer Network Systems

TCP

Reliable Transmission

Transport Layer



telnet, ftp, email

TCP, UDP

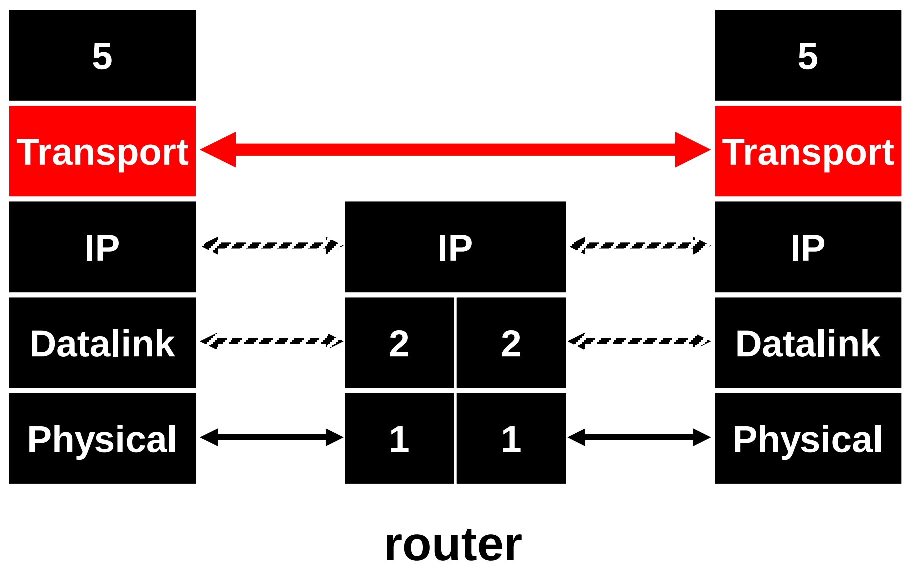
IP

802.3, 802.11

Transport Protocols Concern only End Hosts, not Routers

- Header generated by sender is interpreted only by the destination

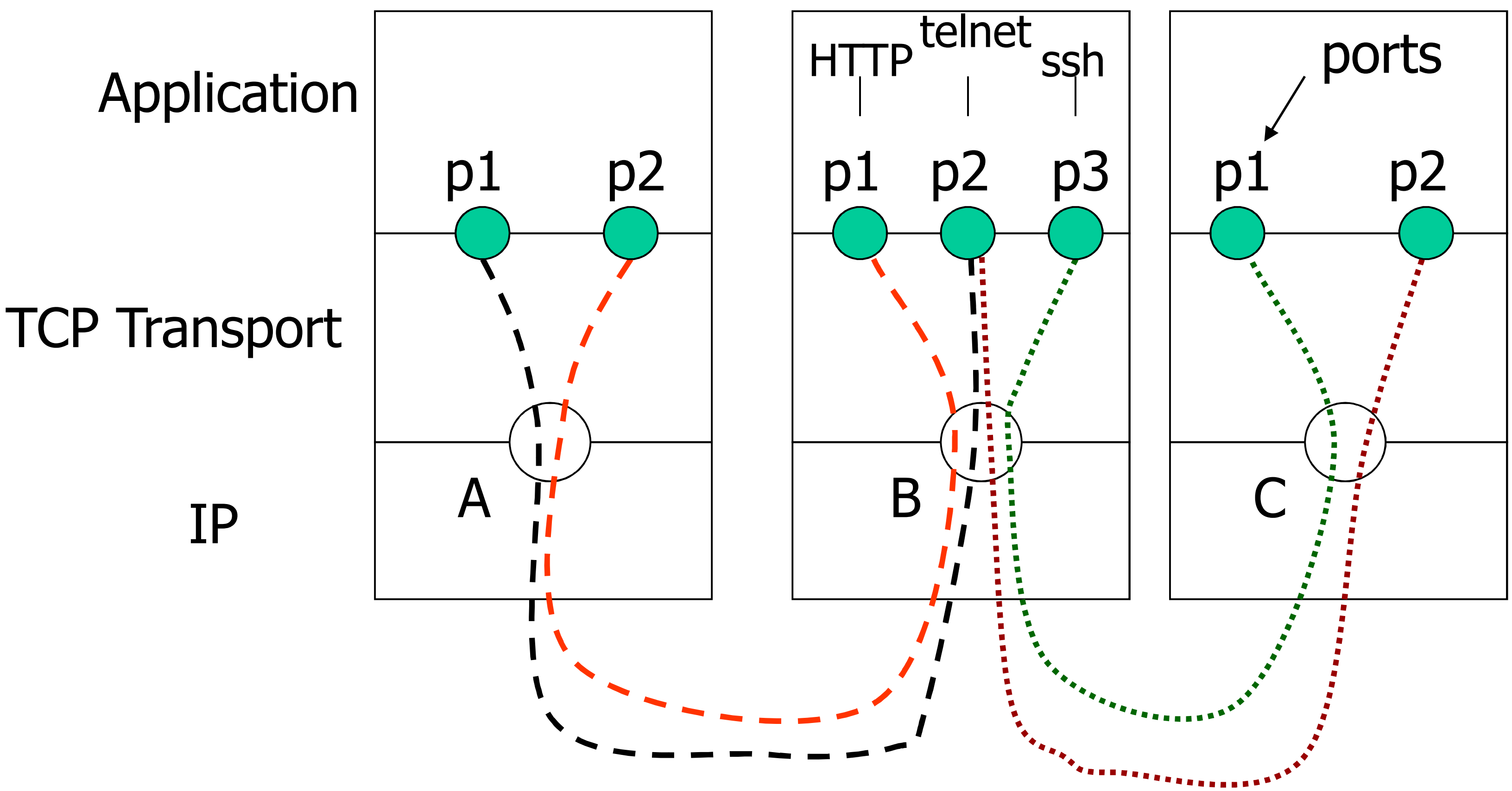
- Routers view transport header as part of the payload



Transport Layer in Internet

- Purpose 1: (De)multiplexing of data streams to different application processes
- Purpose 2: Provide value-added services that many applications want
  - Recall network layer in Internet provides a “Best-effort” service only, transport layer can add value to that
    - Application may want reliability, etc
  - No need to reinvent the wheel each time you write a new application

Using Transport Layer Port Number to (De)multiplex traffic



In TCP, a data stream is identified by a set of numbers:
(Source Address, Destination Address, Source Port, Destination Port)

Popular Transport Protocols

- UDP:
  - Barebones,minimal
  - Does not provide much functionality besides multiplexing
- TCP:
  - Elaborate, lots of additional functionality provided.

User Datagram Protocol (UDP)

- Connectionless datagram
  - Socket: SOCK_DGRAM
- Port number used for (de)multiplexing
  - port numbers = connection/application endpoint
- Adds end-to-end reliability through optional checksum
  - protects against data corruption errors between source and destination (links, switches/routers, bus)
  - does not protect against packet loss, duplication or reordering

0 16 32

Source Port	**Dest. Port**
Length	**Checksum**

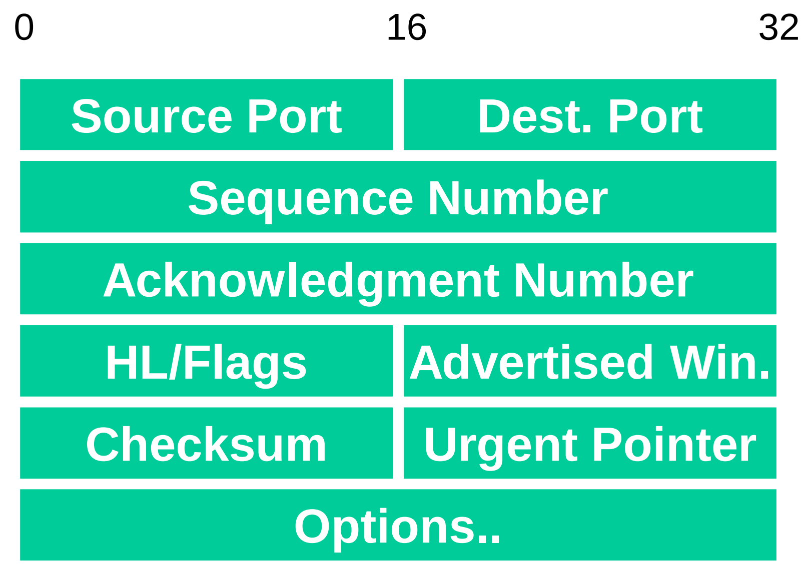
Using UDP

- Custom protocols/applications can be implemented on top of UDP
  - use the port addressing provided by UDP
  - implement own reliability, flow control, ordering, congestion control as it sees fit
- Examples:
  - remote procedure call
  - Multimedia streaming (real time protocol)
  - distributed computing communication libraries

Transmission Control Protocol (TCP)

- Reliable bidirectional in-order byte stream
  - Socket: SOCK_STREAM
- Lots of functionality
- Connection establishment.
  - Logical end-to-end connection, connection state to optimize performance
- Error control
  - Hide unreliability of the network layer from applications
  - Many types of errors: corruption, loss, duplication, reordering.
- End-to-end flow control and congestion control
  - Avoid flooding the receiver and network.

TCP Header

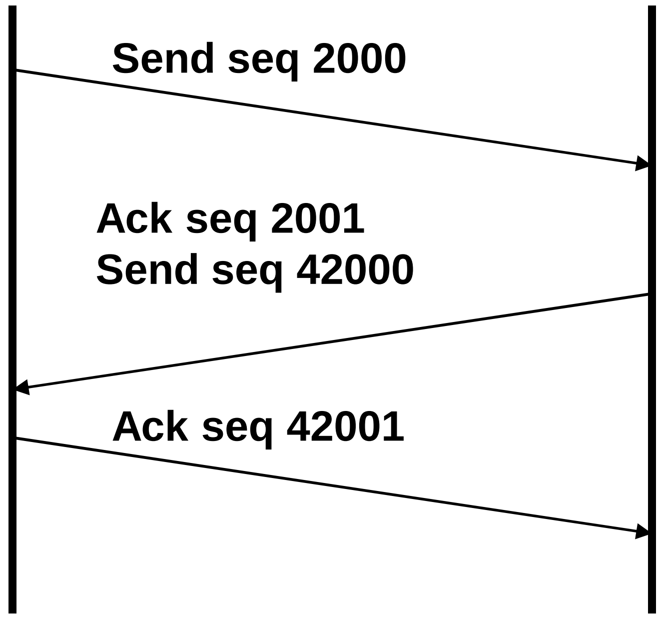


- 20 bytes total
- Sequence Number, ACK, Advertised window: relate to TCP functionality for achieving reliable delivery.
- Sequence Number is for forward direction, ACK for reverse direction.
- HL: Specifies Header Length
- Flags: 6 flags in all
- Urgent pointer: not common TCP usage, used to signal certain data is “out-of-band” and must be processed immediately

Important TCP Flags

- SYN: Synchronize
  - Used when setting up connection
- FIN: Finish
  - Used when tearing down connection
- ACK
  - Acknowledging received data
- RESET:
  - Receiver wants to abort connection, as it received unexpected segment.
- Push and Urgent flags:
  - Not as commonly used.
  - Signify receiving process must be notified, or out-of-band data

Bidirectional Communication



- Each Side of Connection can Send *and* Receive
- What this Means
  - Maintain different sequence numbers for each direction
  - Single segment can contain new data for one direction, plus acknowledgement for other
    - But some contain only data & others only acknowledgement

Ongoing Communication

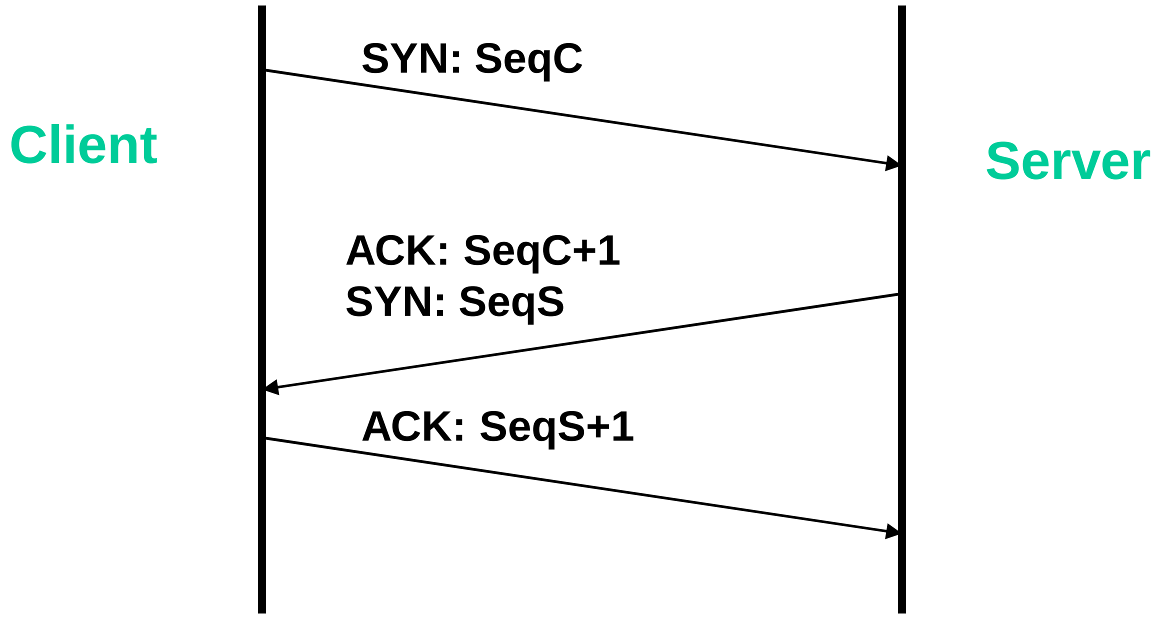
- Bidirectional Communication
  - Each side acts as sender & receiver
  - Every message contains acknowledgement of received sequence
    - Even if no new data have been received
  - Every message advertises window size
    - Size of its receiving window
  - Every message contains sent sequence number
    - Even if no new data being sent
- When Does Sender Actually Send Message?
  - When sending buffer contains at least max. segment size (-header sizes) bytes
  - When application tells it
    - Set PUSH flag for last segment sent
  - When timer expires

Connection Setup

- Why need connection setup?

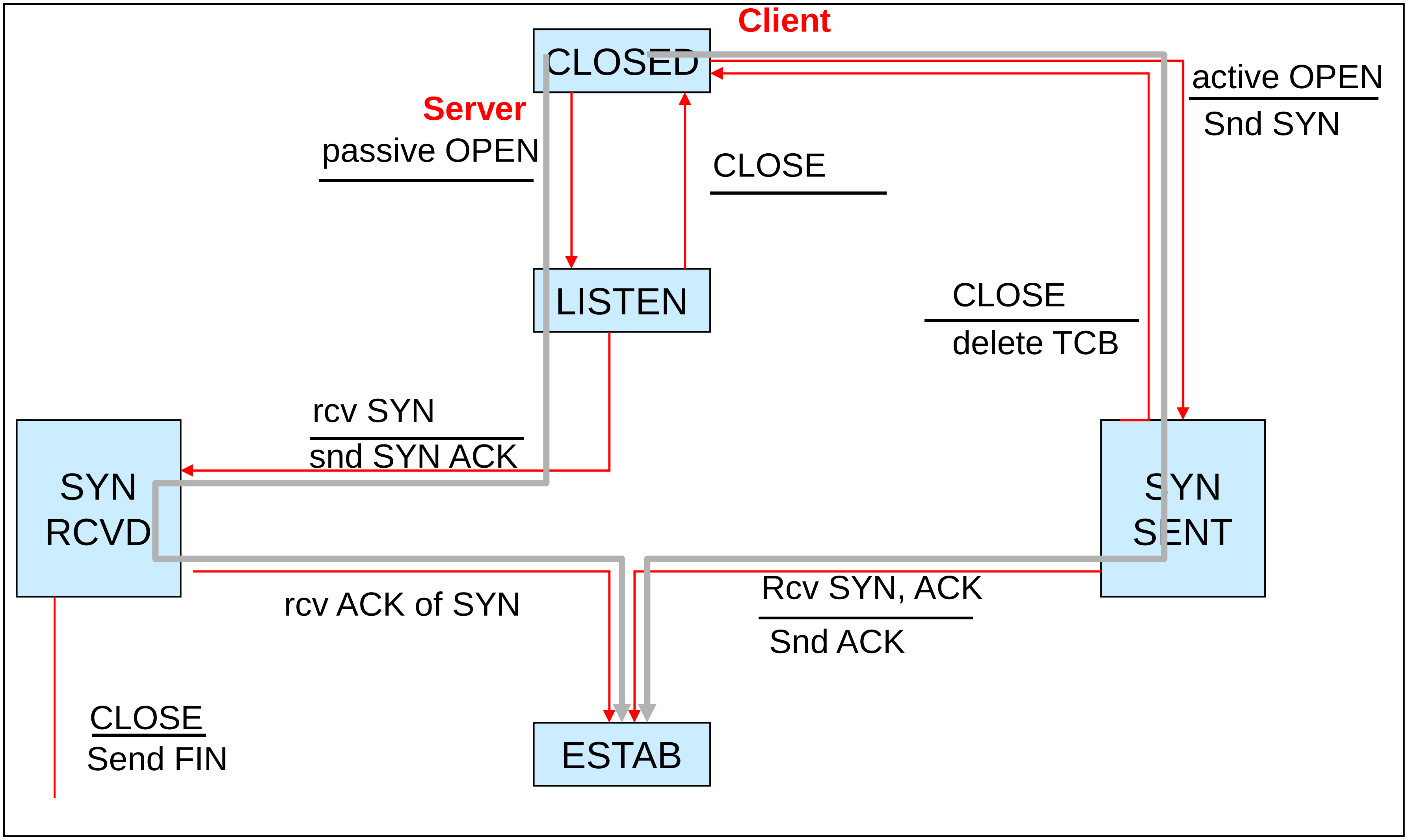
- Mainly to agree on starting sequence numbers
  - Starting sequence number is randomly chosen
  - Reason, to reduce the chance that sequence numbers of old and new connections from overlapping

Establishing Connection



- Three-Way Handshake
  - Each side notifies other of starting sequence number it will use for sending
  - Each side acknowledges other's sequence number
    - SYN-ACK: Acknowledge sequence number + 1
  - Can combine second SYN with first ACK

TCP State Diagram: Connection Setup

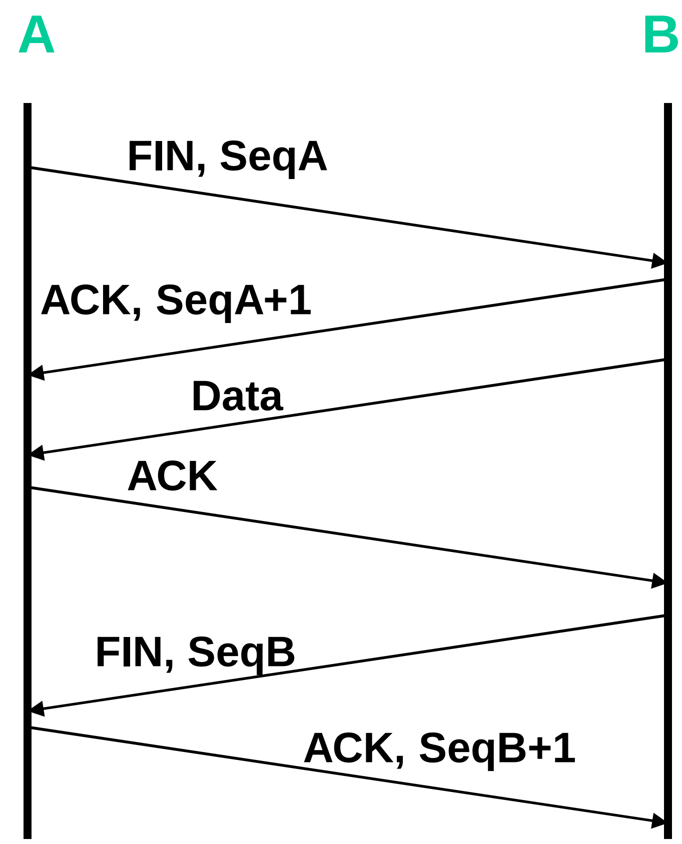


SYN-Flood Attacks

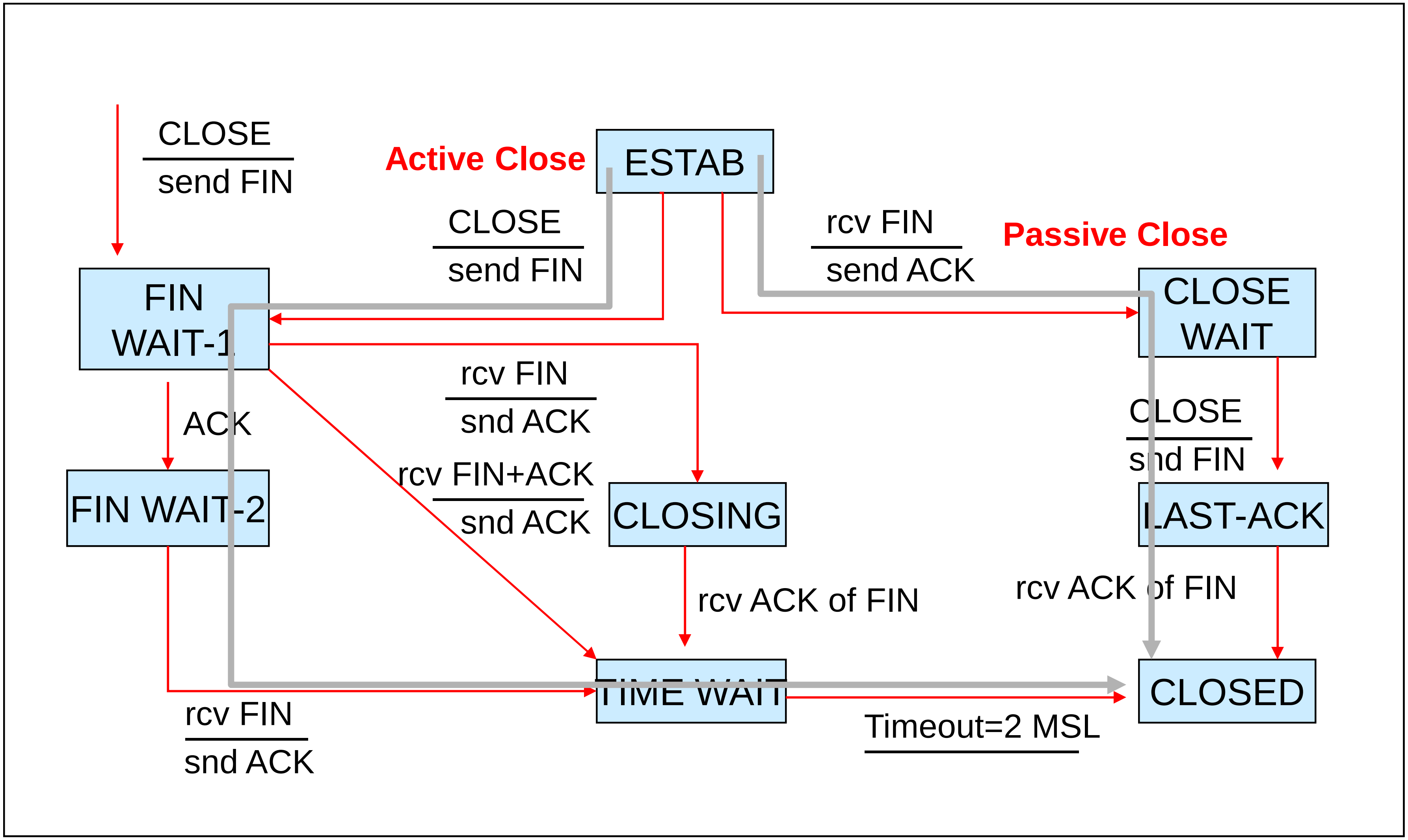
- Exploit 3-way handshake
- Attacker spoofs IP addresses, sends a large number of SYNs to the destination.
- Creates “partial unopened” state at the server
- Server sends SYN-ACK to (spoofed) addresses, to which no response is received.
- Rate at which SYNs sent much faster than rate at which server eliminates partial state.

Tearing Down Connection

- Either Side Can Initiate Tear Down
  - Send FIN signal
  - “I’m not going to send any more data”
- Other Side Can Continue Sending Data
  - Half open connection
  - Must continue to acknowledge
- Acknowledging FIN
  - Acknowledge last sequence number + 1



State Diagram: Connection Tear-down

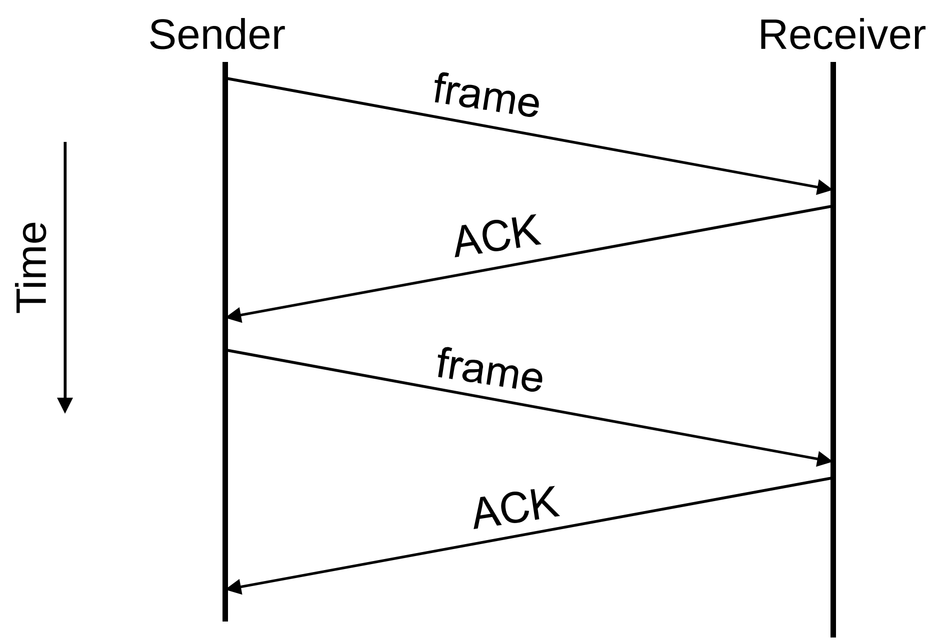


Reliable Transmission

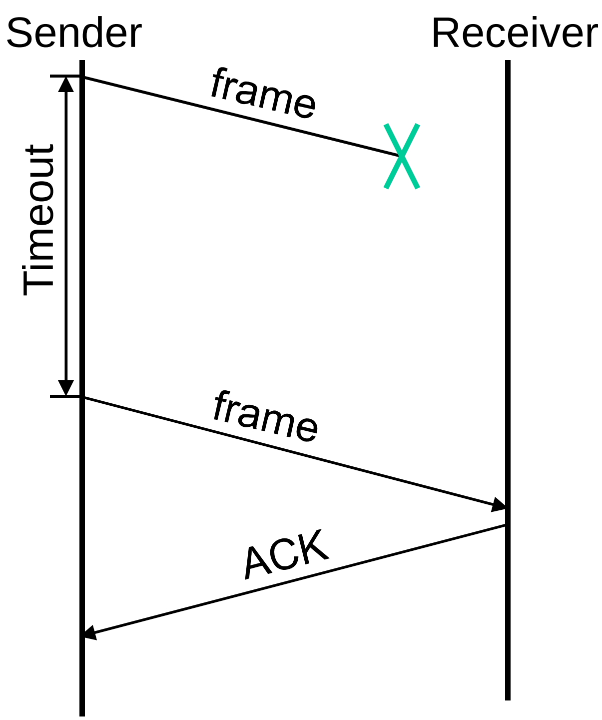
- Use two basic techniques:
  - Acknowledgements (ACKs)
  - Timeouts
- Two examples:
  - Stop-and-go
  - Sliding window

Stop-and-Go

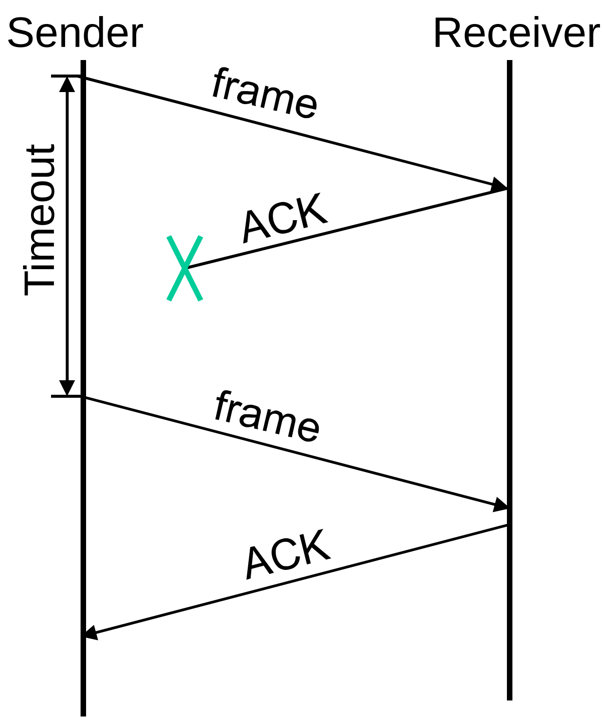
- Receiver: send an acknowledge (ACK) back to the sender upon receiving a packet (frame)
- Sender: excepting first packet, send a packet only upon receiving the ACK for the previous packet



What Can Go Wrong?

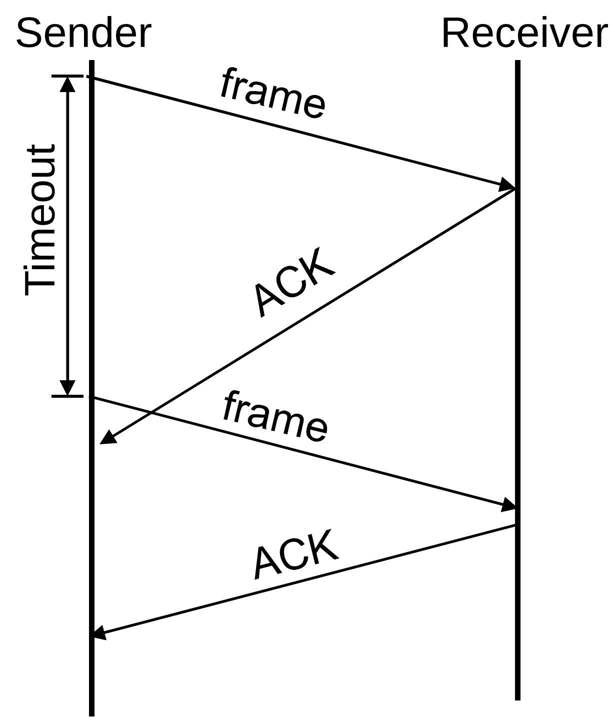


Frame lost → resent it on Timeout



ACK lost → resent packet

Need a mechanisms to detect duplicate packet

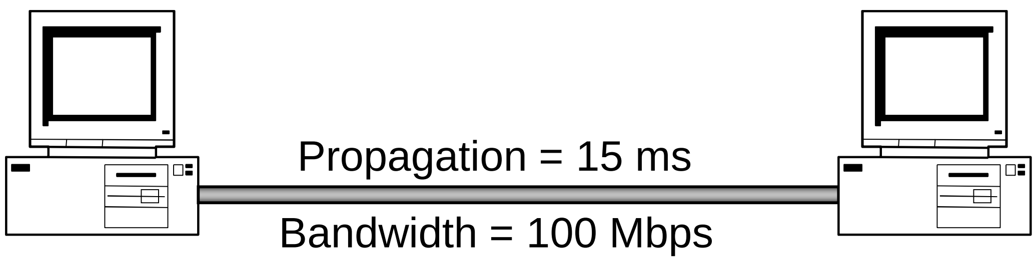


ACK delayed → resent packet

Need at least one bit to differentiate between ACK for current and previous packet

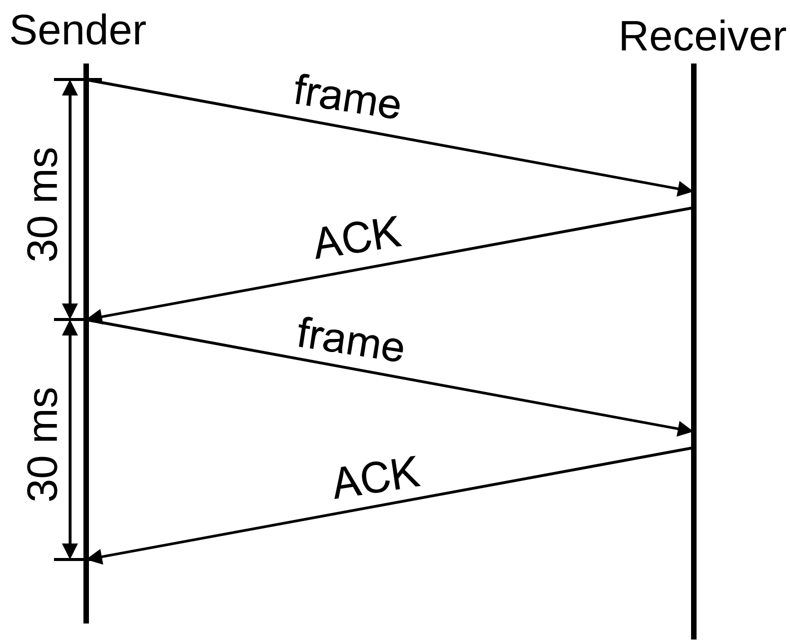
Stop-and-Go Disadvantage

- May lead to inefficient link utilization
- Example: assume
  - One-way propagation = 15 ms
  - Bandwidth = 100 Mbps
  - Packet size = 1000 bytes → transmit = $(8*1000)/10^8$ = 0.08ms
  - Neglect queue delay → Latency = approx. 15 ms; RTT = 30 ms

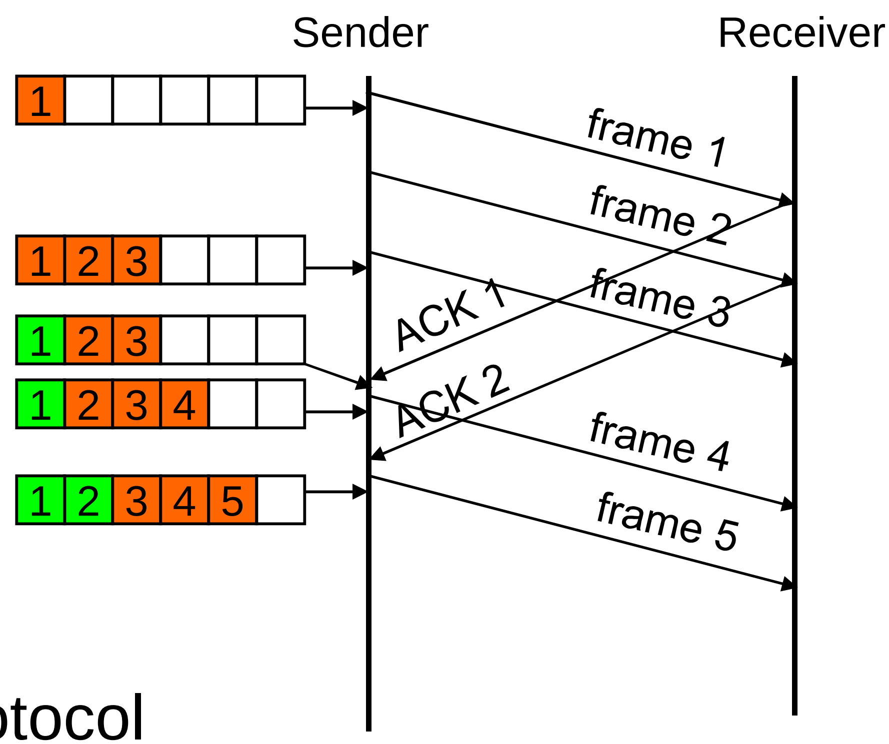


Stop-and-Go Disadvantage (cont'd)

- Send a message every 30 ms → Throughput = (8*1000)/0.03 = 0.2666 Mbps
- Thus, the protocol uses less than 0.3% of the link capacity!



Sliding Window Approach



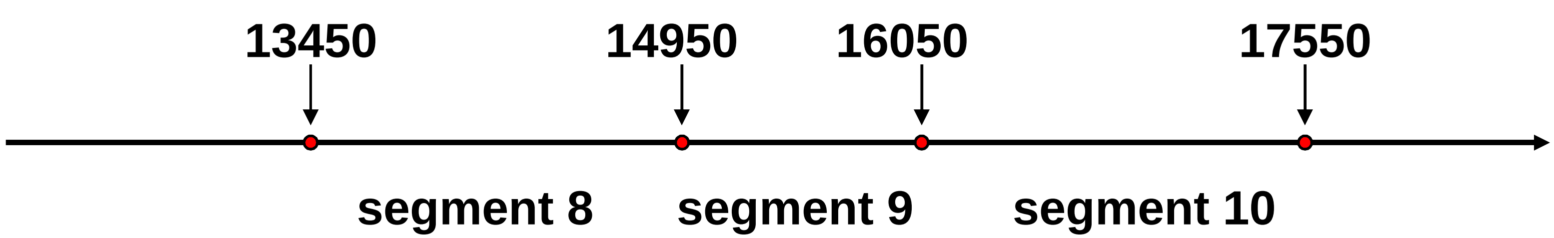
Sliding window protocol

Send multiple packets without waiting for ACK.

Sliding window size: Max number of packets that can be sent without ACK being received (3 in figure)

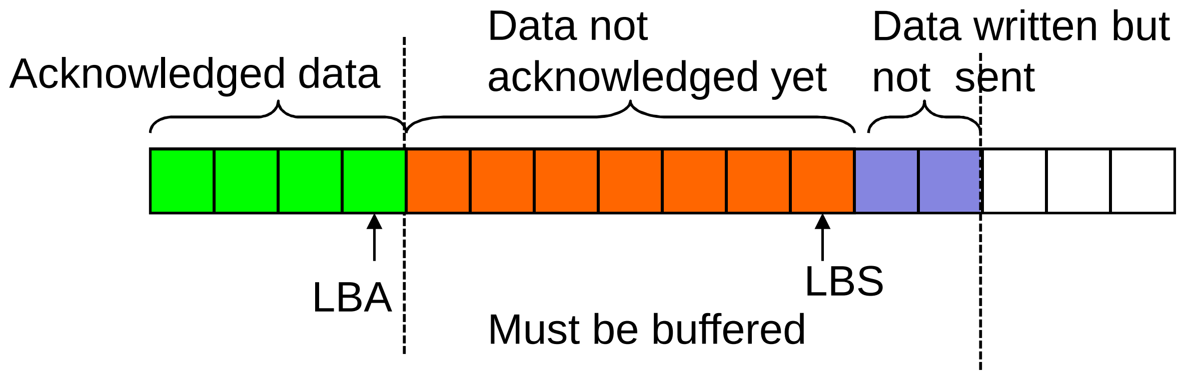
Sequence Numbers in TCP

- Each byte in byte stream is numbered.
  - 32 bit value
- TCP breaks up the byte stream in segments
- Each segment has a sequence number.
  - Indicates where it fits in the byte stream



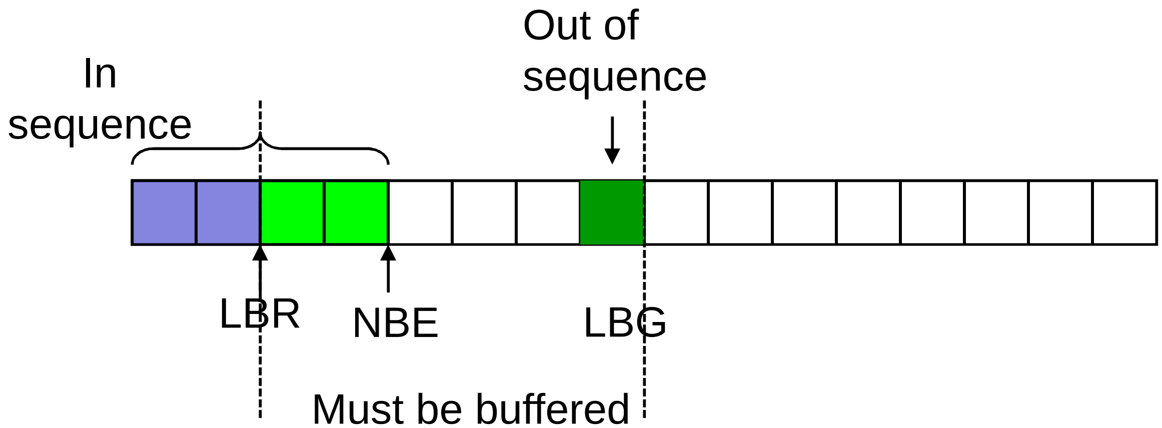
Sliding Window Protocol: Sender

- TCP: operates at a byte stream level (rather than packet level)
- Sender maintains a window of sequence numbers
  - SWS (sender window size) – maximum data that can be sent without receiving an ACK
  - LBA (last byte acknowledged)
  - LBS (last byte sent)
- TCP sender side socket buffer:
  - Data sent but not acknowledged
  - Data written by application but not sent yet



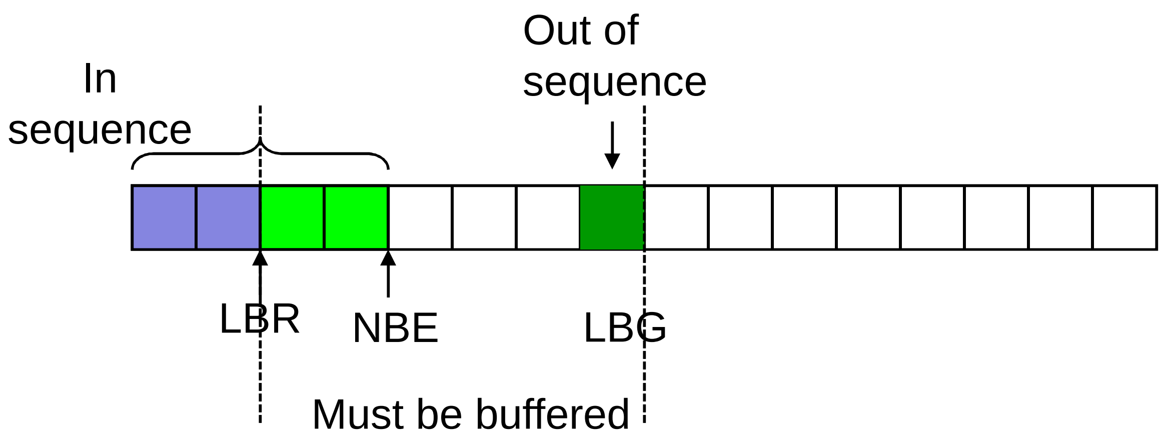
Sliding Window Protocol: Receiver

- TCP maintains a receive socket buffer
  - Stores data that arrived out-of-order (cannot be given to application yet)
  - Also stores data that arrived in-order but not yet read by application (slow process)
- Receiver maintains a window of sequence numbers
  - NBE (next byte expected – all previous bytes received in sequence
  - LBR (last byte read by application)
  - LBG (last byte got by receiver)



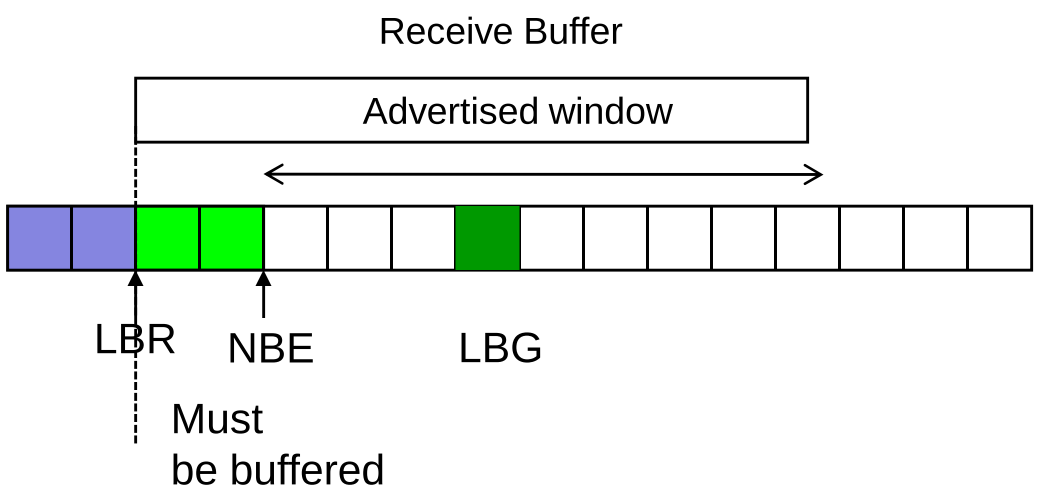
Sliding Window Protocol: Receiver

- If incoming byte < NBE
  - Discard packet
- Else
  - Accept packet (provided it fits in the buffer)
  - ACK largest byte that all previous bytes were received



Flow Control

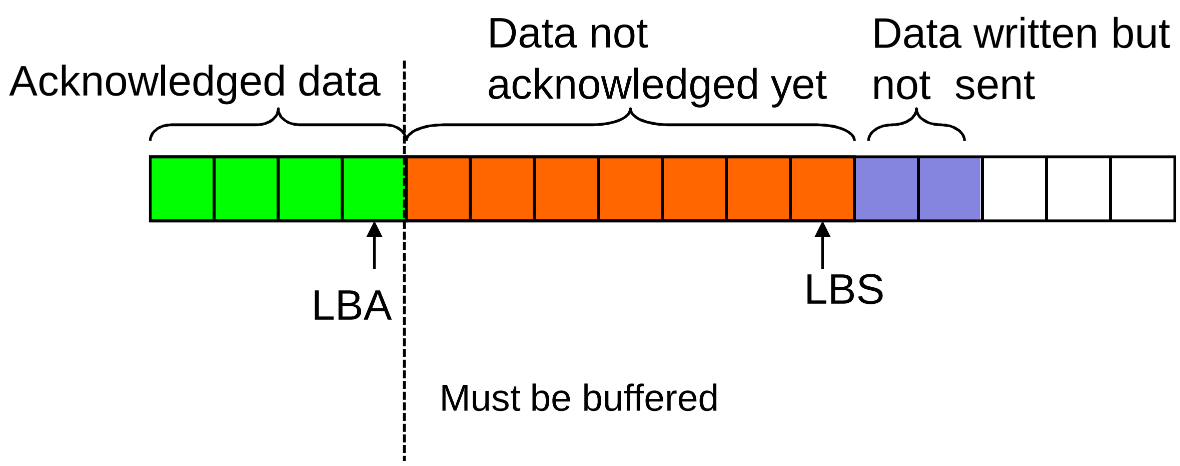
- Receiver throttles the sender by advertising a window no larger than the amount of data it can buffer
- Advertised Window depends on
  - Size of receive buffer
  - How fast receiver process can read the bytes from the buffer
- TCP sending window impacted by advertised window



Flow Control (sender side)

- Receiver throttles the sender by advertising a window no larger than the amount of data it can buffer
- Effective window at sender (limits how much it can send)

  Advertised window – (outstanding unacknowledged data)
- Sending process can write to send-side TCP buffer only if space in sender side buffer – otherwise will block



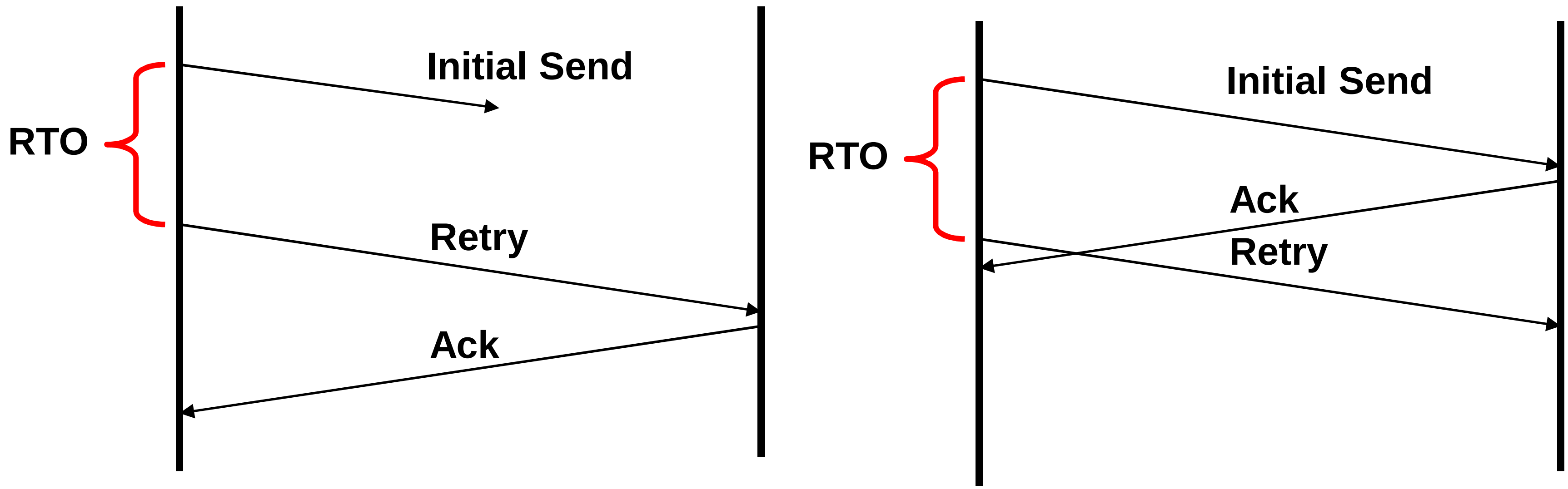
Buffer sizes

- Size of TCP socket buffers at sender and receiver side critical
- Would like this to be Bandwidth * Delay product of network, to ensure the pipe can be full.
- Must be tuned carefully for individual path.
- Traditionally, OSs used default socket buffer sizes that must be overriden by application programmer/end-user
  - "Tuning" in manual fashion tricky/hard.
- Research efforts at "auto-tuning" TCP buffers:
  - **Automatic TCP Buffer Tuning**
    Jeffrey Semke, Matthew Mathis, and Jamshid Mahdavi, SIGCOMM 1998
- More recently finding its way into some Operating Systems.

Sequence Number Wraparound

- TCP sequence numbers: 32 bits.
- Could wraparound
- Assuming packets remain in the network at most 120s, wrap around is a problem only for bandwidth above 286 Mbps
  - OK for most common networks
  - An issue for high-speed networks (e.g. special purpose scientific networks)
  - 120s: “Maximum segment lifetime”: just a conservative estimate of how long a packet may linger in the network: not actual value.

Setting Retransmission Timeout (RTO)



Detect dropped packet

RTO too short

- Time between sending & resending segment

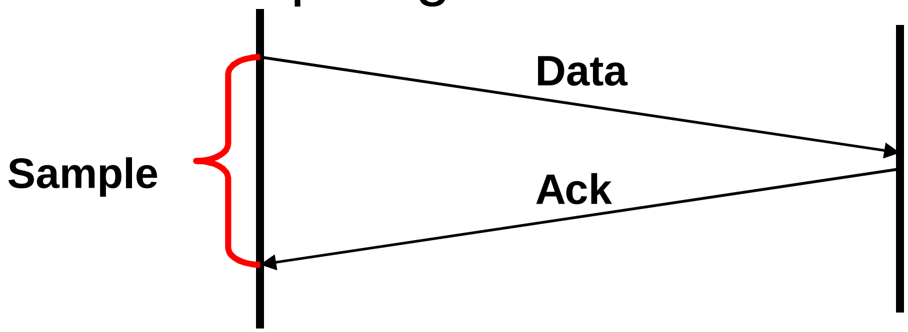
- Challenge
  - Too long: Add latency to communication when packets dropped
  - Too short: Send too many duplicate packets
  - General principle: Must be > 1 Round Trip Time (RTT)

Setting Timeouts

- Retransmission time-out should be set based on round-trip delay
- But round-trip delay different for each path!
- Must estimate RTT dynamically

Round-trip Time Estimation

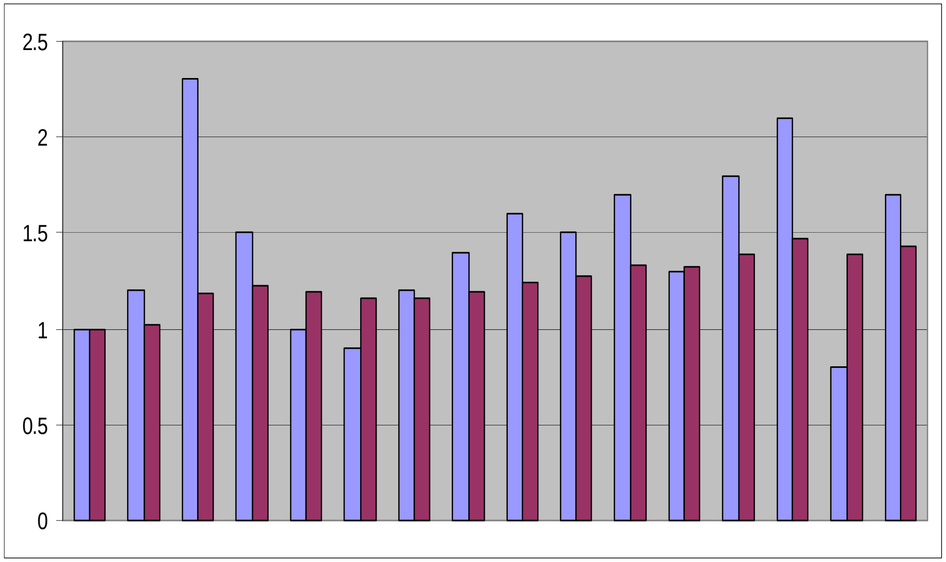
- Every Data/Ack pair gives new RTT estimate



- Can Get Lots of Short-Term Fluctuations

Original TCP Round-trip Estimator

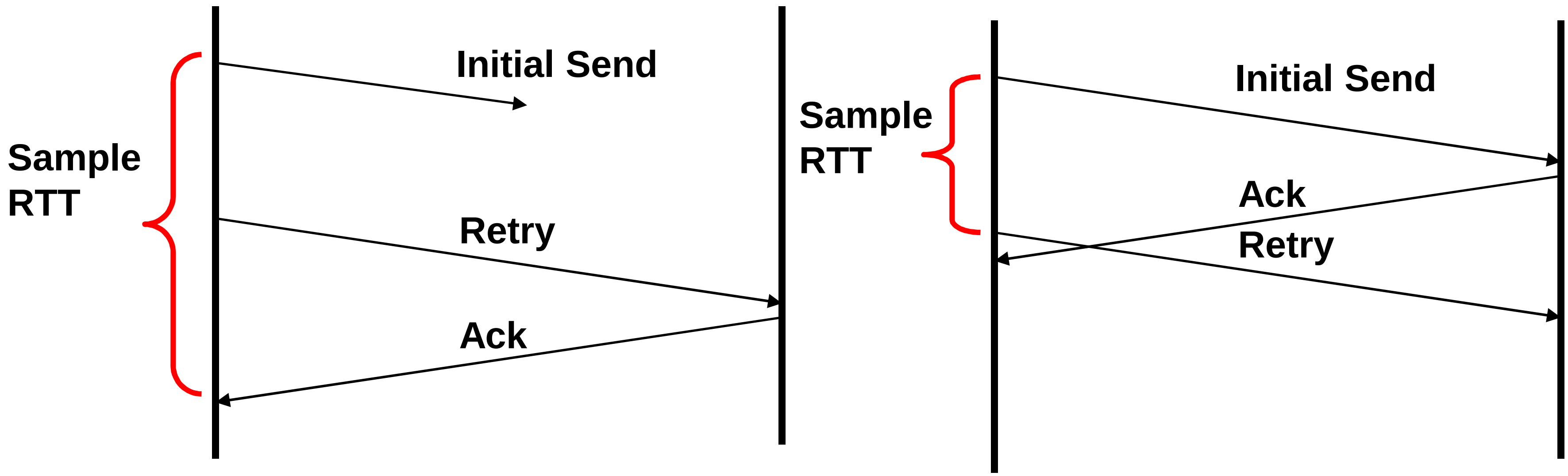
- Round trip times exponentially averaged:
  - New RTT = α (old RTT) + (1 - α) (new sample)
  - Recommended value for α: 0.8 - 0.9
    - 0.875 for most TCP's



- Retransmit timer set to (b * RTT), where b = 2
  - Every time timer expires, RTO exponentially backed-off

- Enhanced algorithm used in practice
  - [Jacobson/Karels: see textbook if interested]

Issue



What value of Sample RTT to use?

Solution: TCP simply neglects SampleRTT in such cases.
[Karn/Partridge algorithm]