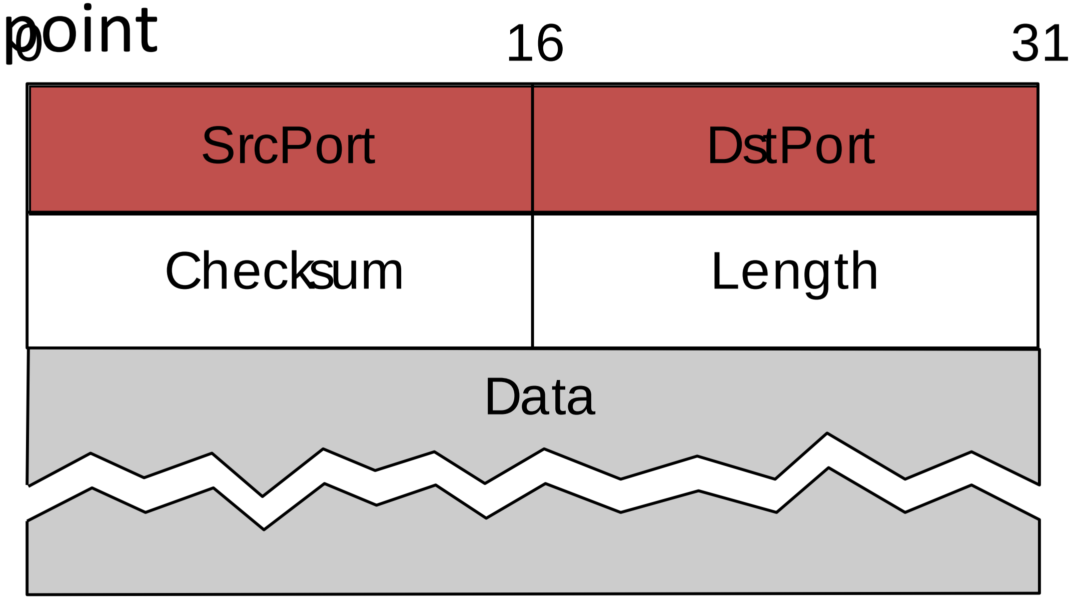
Transport Protocol Design: UDP, TCP

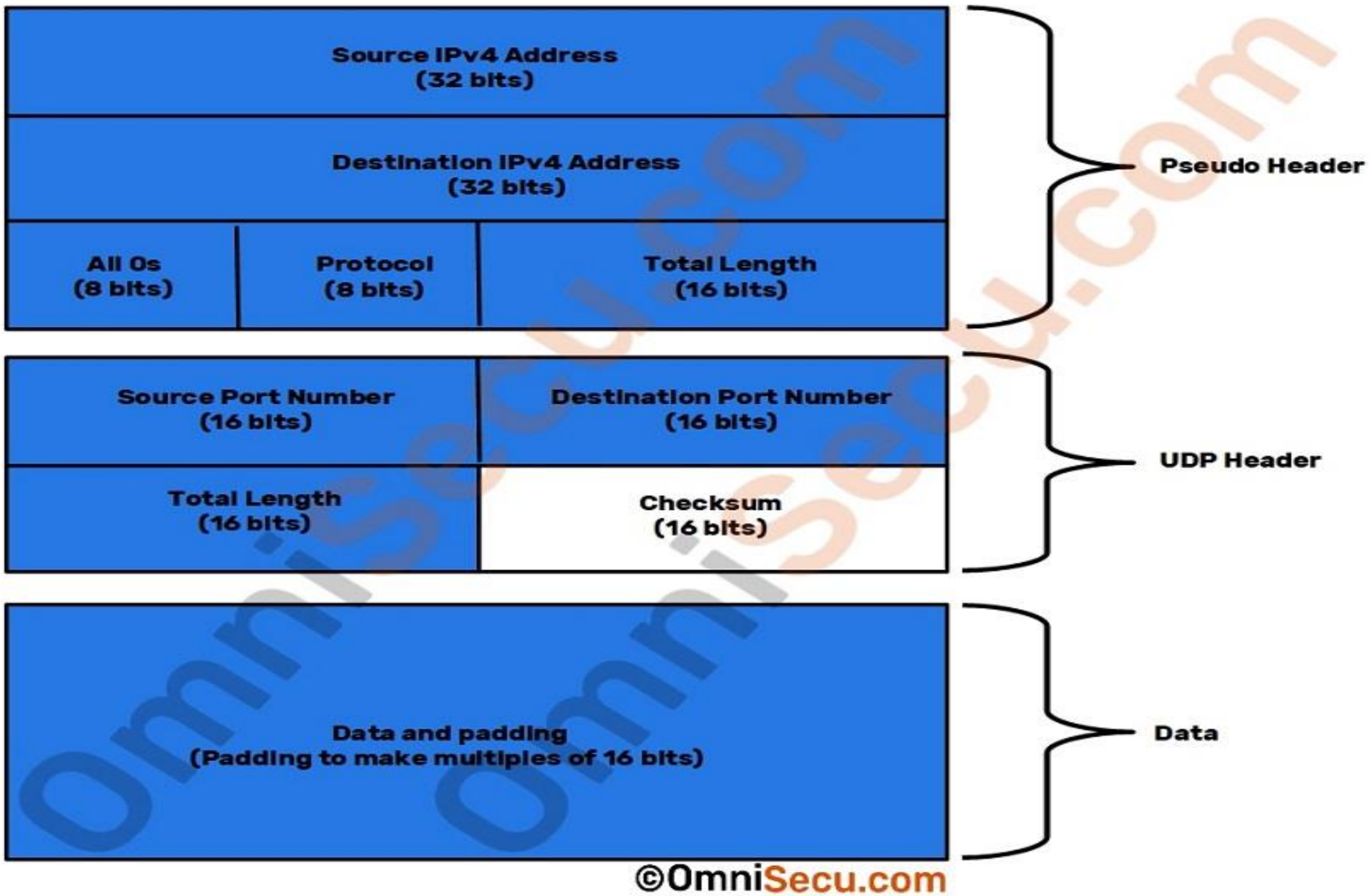
User Datagram Protocol (UDP)

- Provides message delivery between processes
  - Source port filled in by OS as message is sent
  - Destination port identifies UDP delivery queue at endpoint



UDP Checksum

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



Introduction to TCP

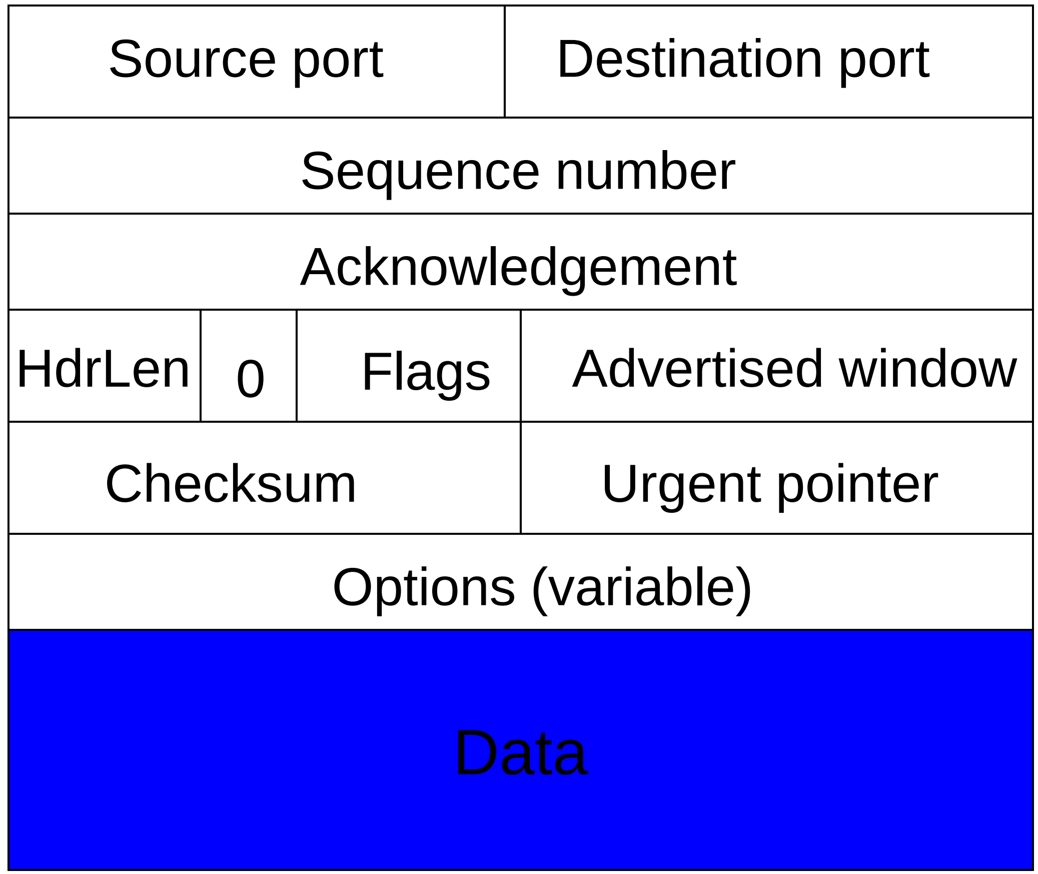
- Communication abstraction:
  - Connection Oriented Protocol
  - Reliable and Ordered data delivery
  - Point-to-point
  - Byte-stream
  - Full duplex
  - Flow and congestion controlled
- Protocol implemented entirely at the ends

Transmission Control Protocol (TCP)

- Connection-oriented
- Byte-stream
  - app writes bytes
  - TCP sends segments
  - app reads bytes
- Full duplex
- Flow control: keep sender from overrunning receiver
- Congestion control: keep sender from overrunning network

TCP Header

Flags:
1. SYN
2. FIN
3. RESET
4. PUSH
5. URG
6. ACK

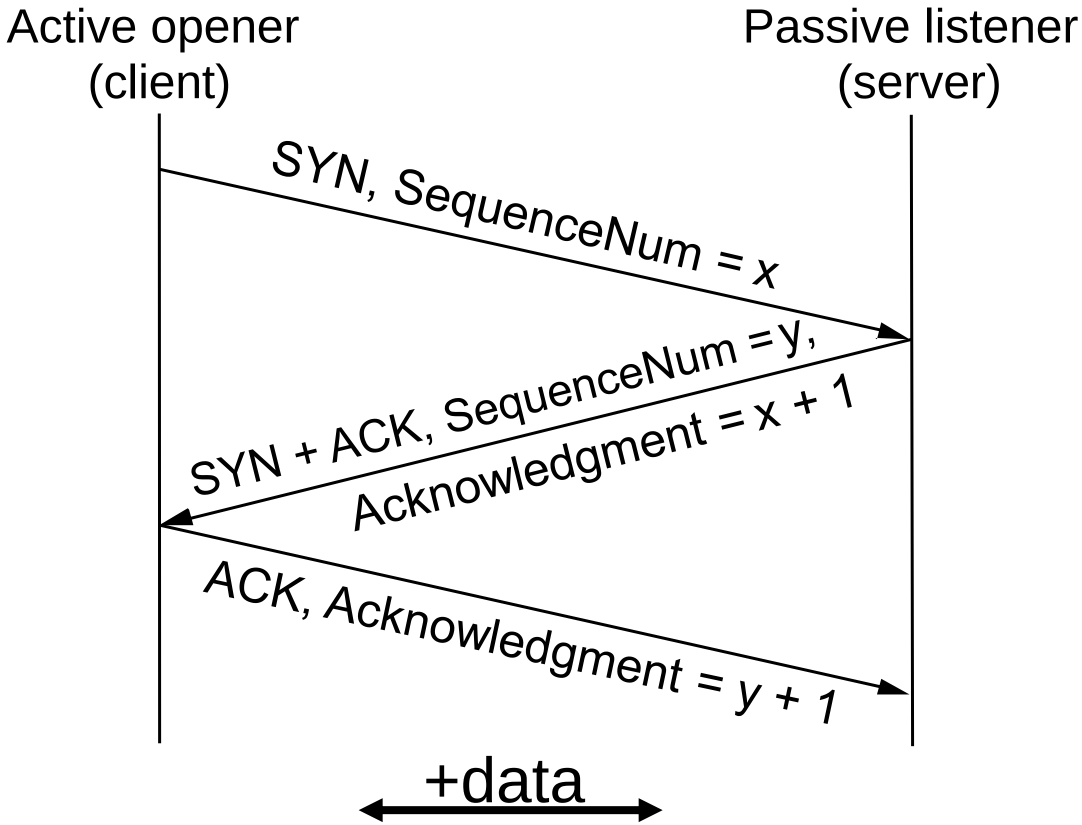


TCP Header Fields

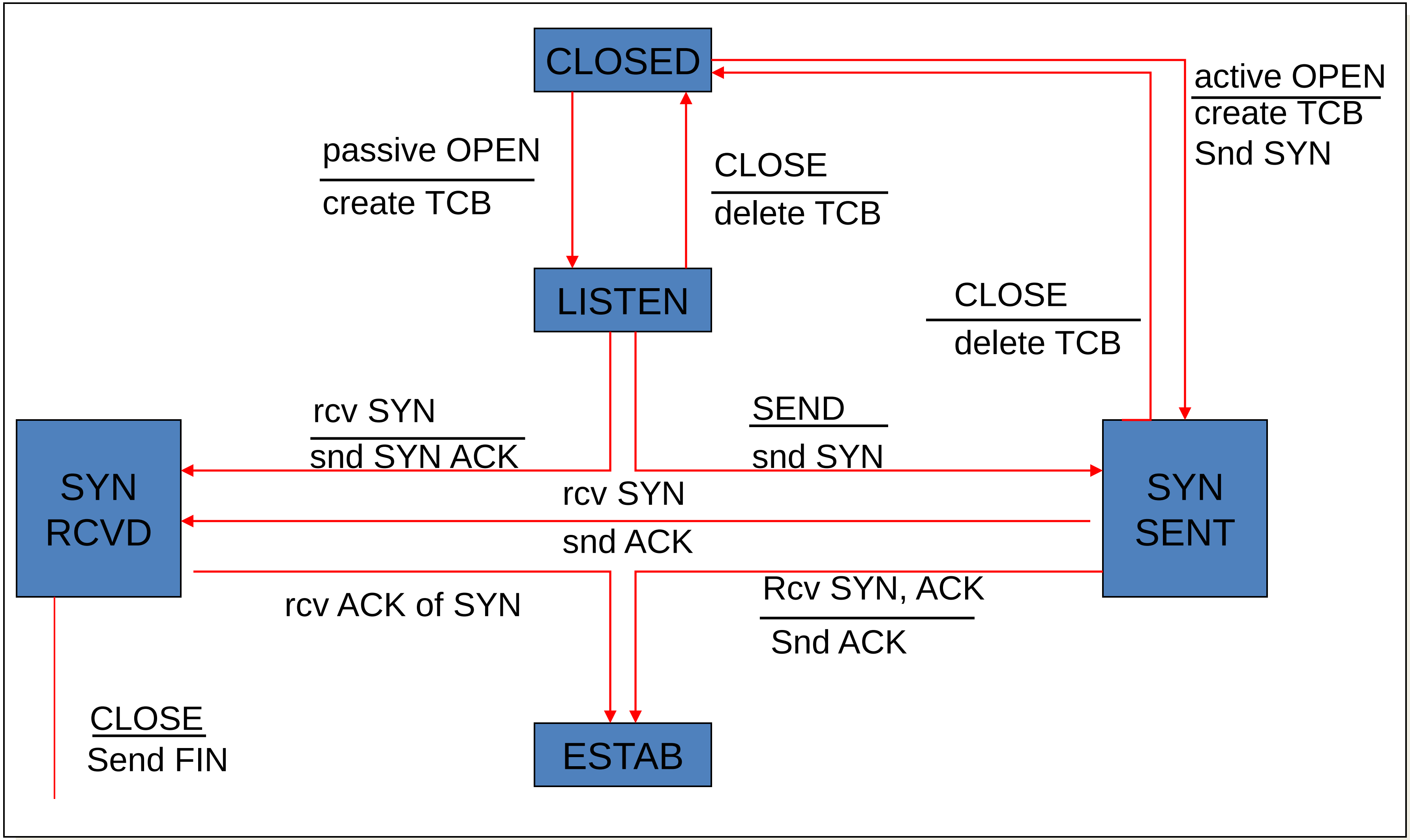
- Each connection identified with 4-tuple:
  - (SrcPort, SrcIPAddr, DsrPort, DstIPAddr)
- Sliding window + flow control
  - acknowledgment, SequenceNum, AdvertisedWinow
- Flags
  - SYN, FIN, RESET, PUSH, URG, ACK
- Checksum
  - pseudo header + TCP header + data

Three-Way Handshake

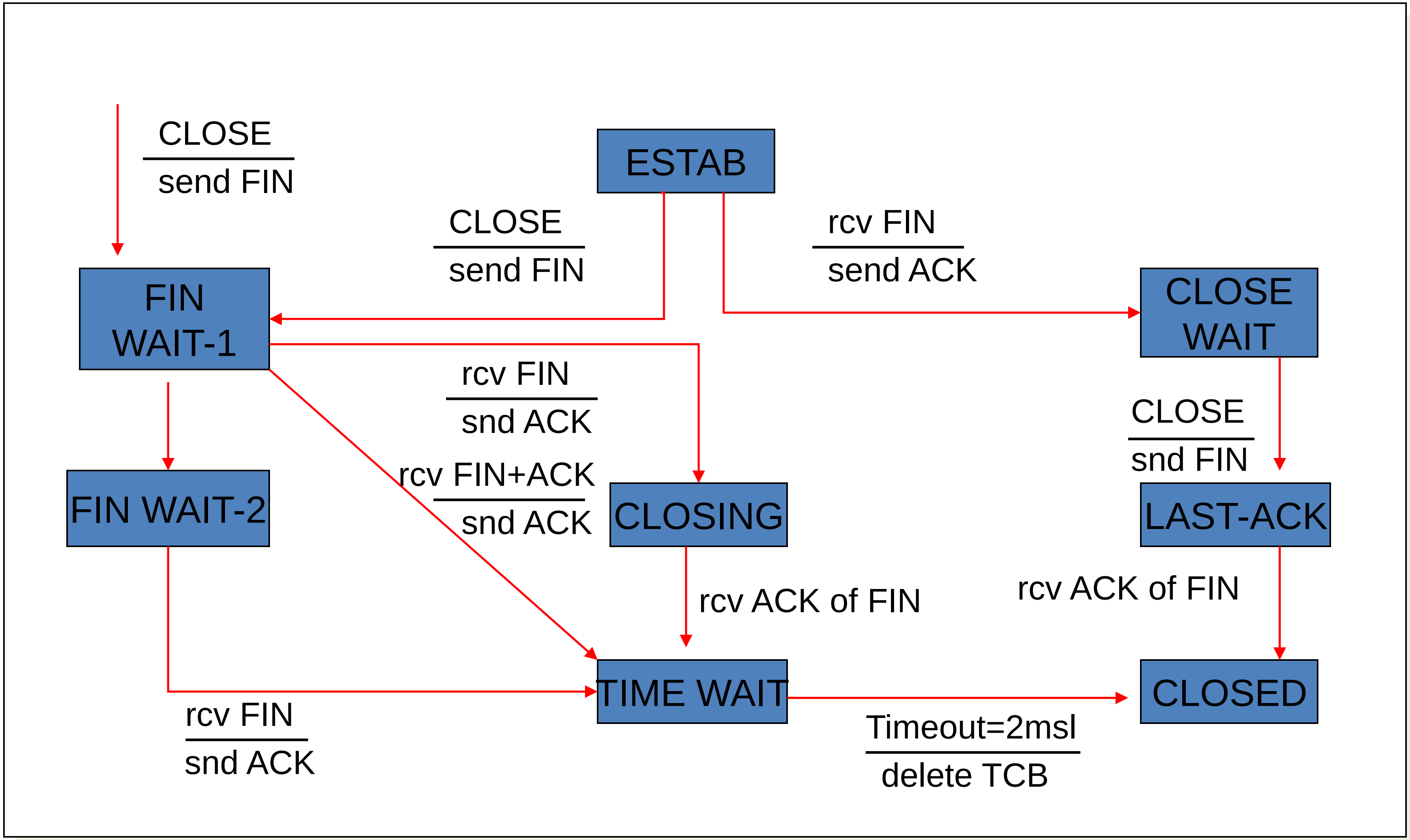
- Opens both directions for transfer



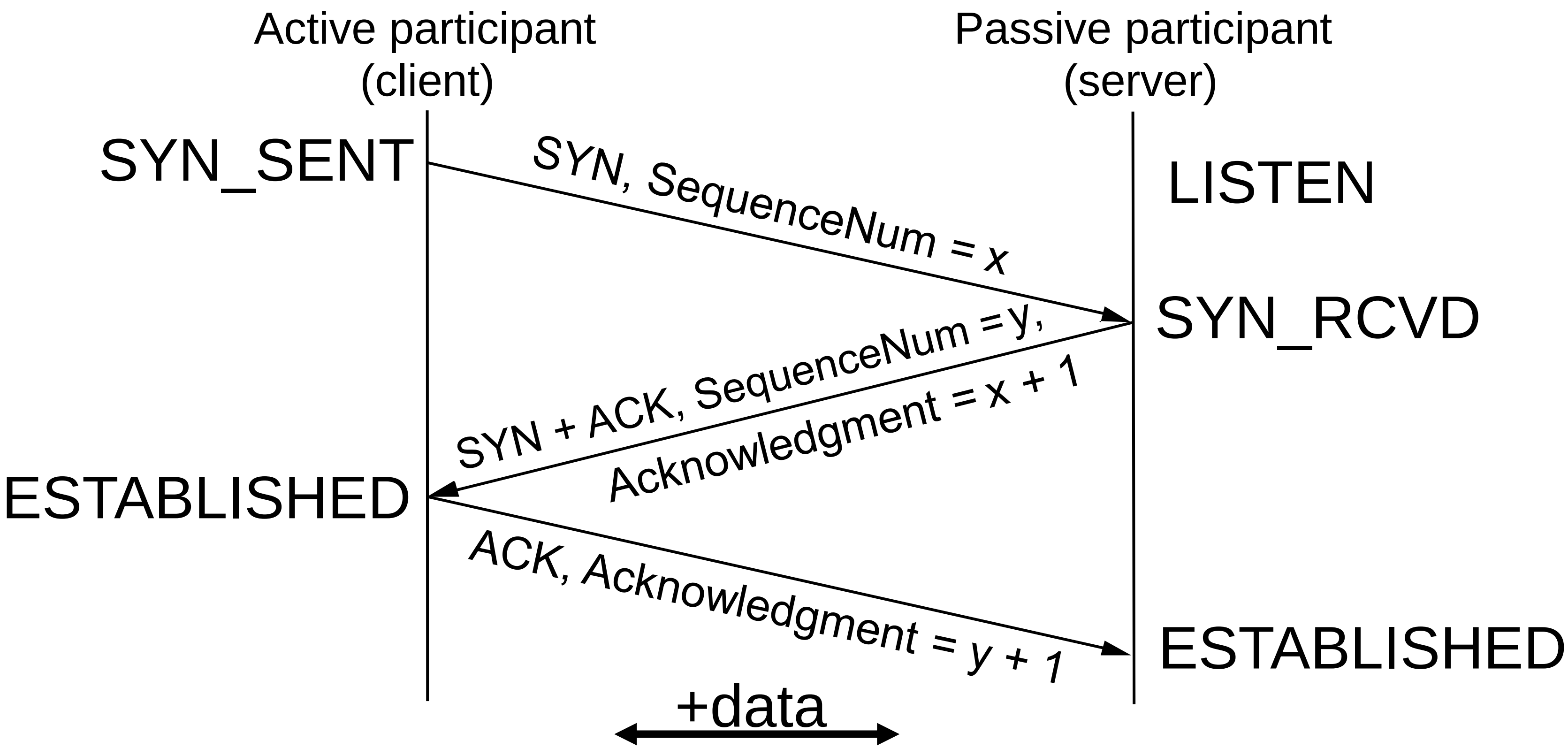
TCP Connection Setup: FSM



TCP Connection Tear-down: FSM



Again, with States



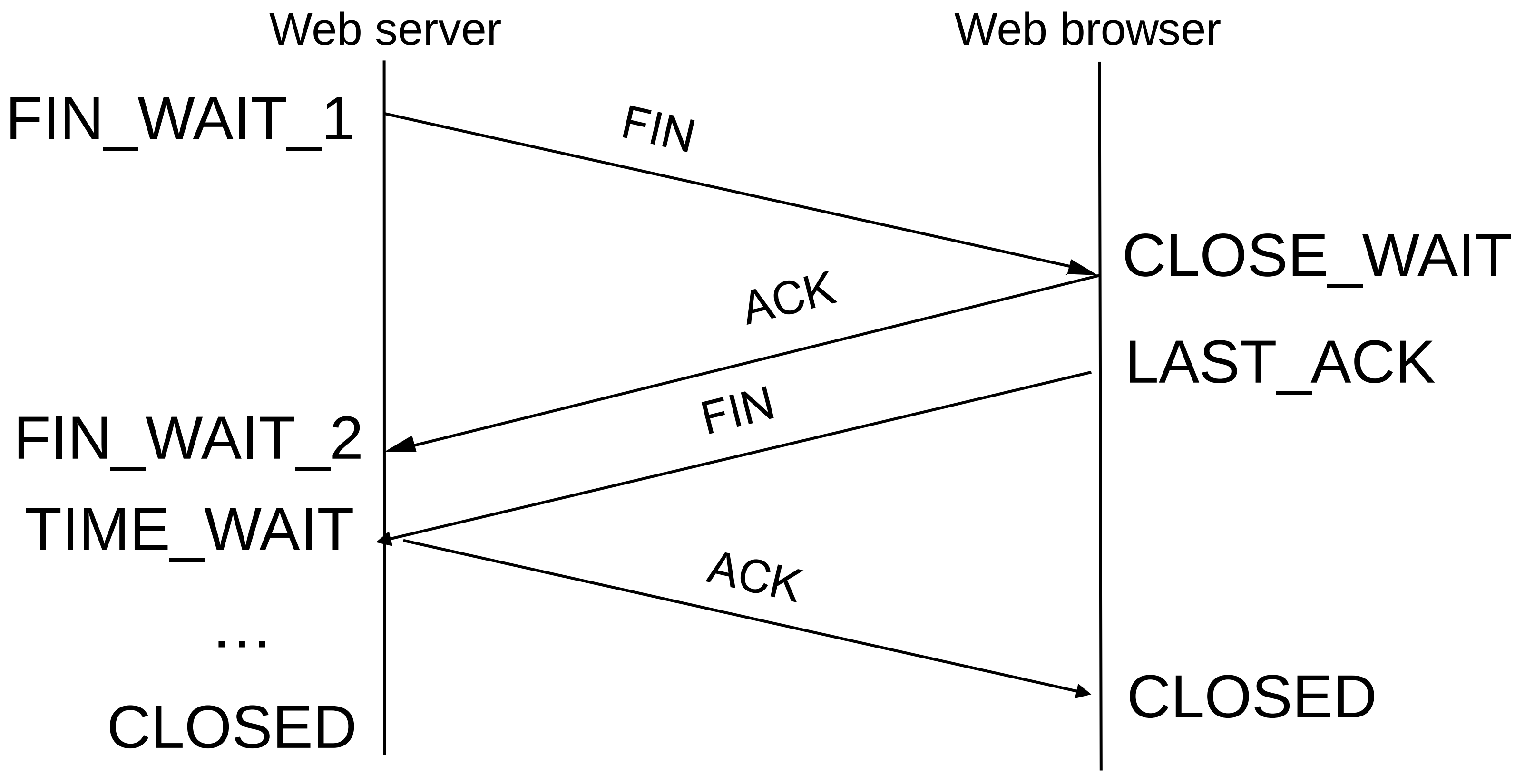
Connection Teardown

- Orderly release by sender and receiver when done
  - Delivers all pending data and “hangs up”

- Cleans up state in sender and receiver

- TCP provides a “symmetric” close
  - both sides shutdown independently

TCP Connection Teardown



The TIME_WAIT State

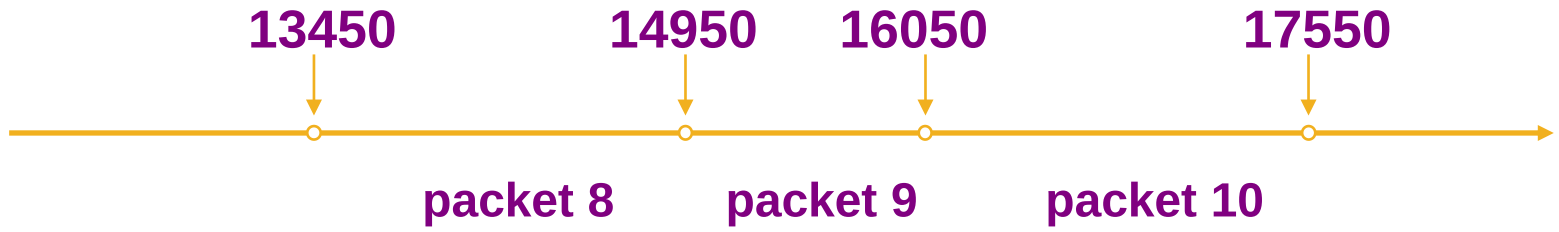
- We wait 2MSL (two times the maximum segment lifetime of 60 seconds) before completing the close

- Why?

- ACK might have been lost and so FIN will be resent
- Could interfere with a subsequent connection

Sequence Number Space

- Each byte in byte stream is numbered.
  - 32 bit value
  - Wraps around
  - Initial values selected at start up time
- TCP breaks up the byte stream in packets.
  - Packet size is limited to the Maximum Segment Size
- Each packet has a sequence number.
  - Indicates where it fits in the byte stream



MSS

- Maximum Segment Size (MSS)
- Largest “chunk” sent between TCPs.
  - Default = 536 bytes. Not negotiated.
  - Announced in connection establishment.
  - Different MSS possible for forward/reverse paths.
  - Does not include TCP header
- What all does this effect?
  - Efficiency
  - Congestion control
  - Retransmission
- Path MTU discovery
  - Why should MTU match MSS?

Silly Window Syndrome

- Problem: (Clark, 1982)
  - If receiver advertises small increases in the receive window then the sender may waste time sending lots of small packets
- Solution
  - Receiver must not advertise small window increases
  - Increase window by min(MSS,RecvBuffer/2)

Nagel’s Algorithm & Delayed Acks

- Small packet problem:
  - Don’t want to send a 41 byte packet for each keystroke
  - How long to wait for more data?
- Solution: Nagel’s algorithm
  - Allow only one outstanding small (not full sized) segment that has not yet been acknowledged

- Batching acknowledgements*:*
  - Delay-ack timer: piggyback ack on reverse traffic if available
  - 200 ms timer will trigger ack if no reverse traffic available

Timeout and RTT Estimation

- Problem:
  - Unlike a physical link, the RTT of a logical link can vary, quite substantially
  - How long should timeout be ?
  - Too long => underutilization
  - Too short => wasteful retransmissions

- Solution: *<u>adaptive timeout:</u> based on a <u>good estimate</u> of <u>maximum current value of RTT</u>*

How to estimate max RTT?

- RTT = prop + queuing delay
  - Queuing delay highly variable
  - So, different samples of RTTs will give different random values of queuing delay

Round Trip Time and Timeout (II)

Q: how to estimate RTT?

- SampleRTT: measured time from segment transmission until ACK receipt
- SampleRTT will vary wildly
  - use several recent measurements, not just current SampleRTT to calculate "AverageRTT
  - **AverageRTT = (1-x)*AverageRTT + x*SampleRTT**
  - Exponential weighted moving average (EWMA)
  - Influence of given sample decreases exponentially fast; x = 0.1 or 0.2

Setting the timeout

Timeout = AverageRTT + 4*Deviation

Deviation = (1-x)*Deviation + x*|SampleRTT- AverageRTT|

Karn's RTT Estimator

- Accounts for *retransmission ambiguity*
- If a segment has been retransmitted:
  - *Don't update RTT estimators during retransmission.*
  - Timer backoff: If timeout, RTO = 2*RTO {exponential backoff}
    - Keep backed off time-out for next packet
  - Reuse RTT estimate only after one successful packet transmission