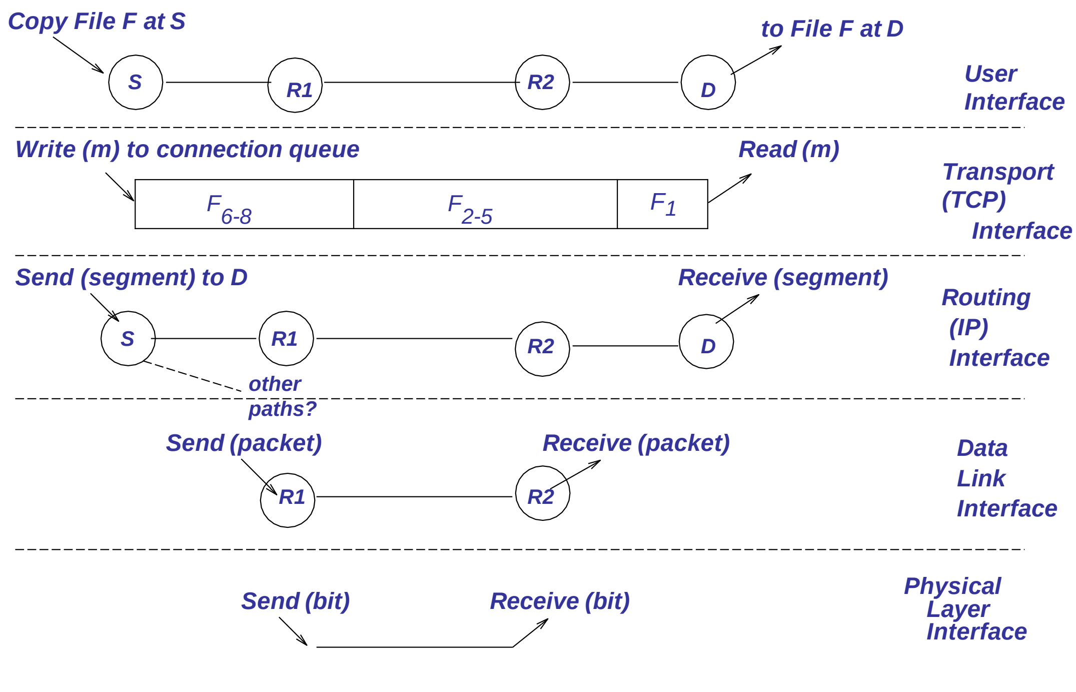
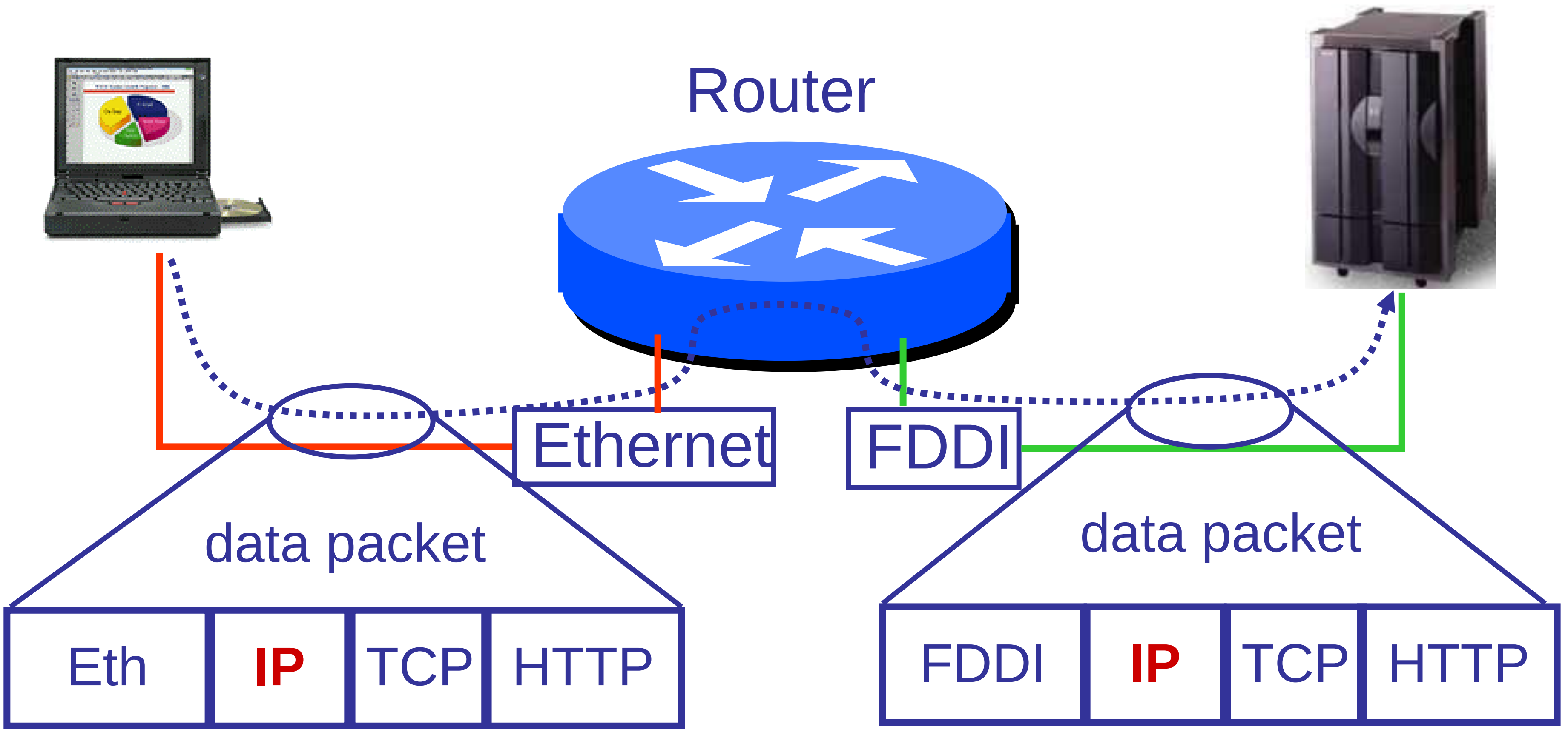
Transition to Transport (TCP)

CSE 118: Computer Networks
George Varghese
(based on slides by Alex Snoeren)

THE IP ABSTRACTIONS:
Each layer provides a service to the layer above it



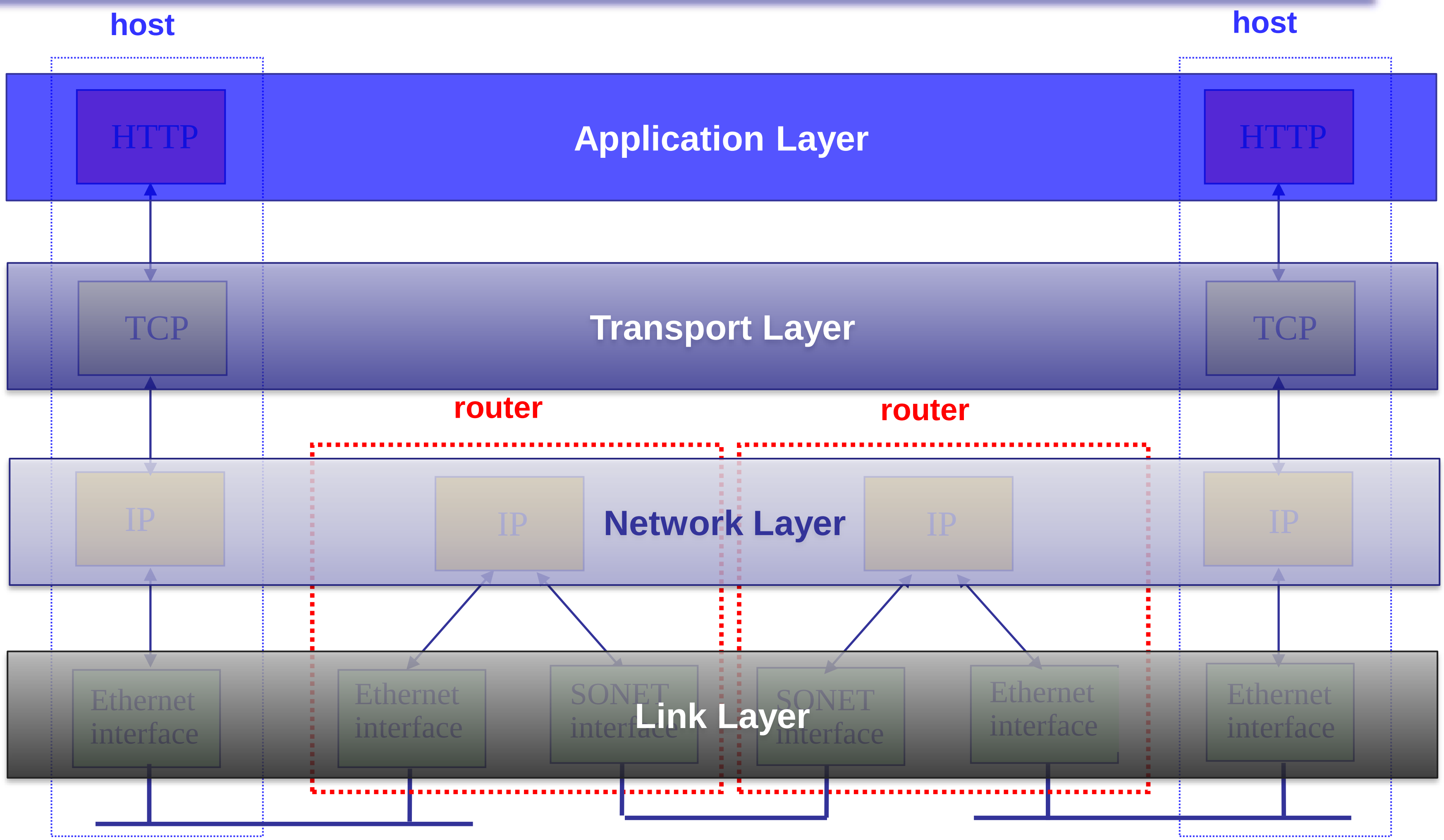
Recall Big Picture



Overview

- Process naming/demultiplexing
- User Datagram Protocol (UDP)
- Transport Control Protocol (TCP)
  - Three-way handshake
  - Flow control

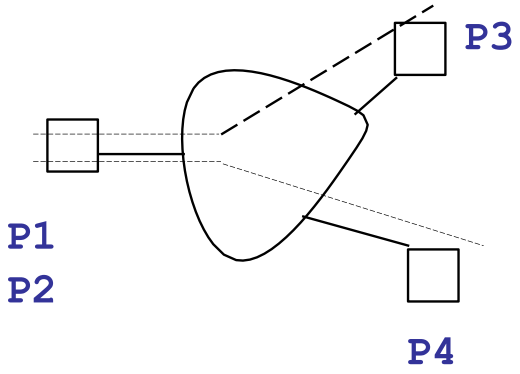
Where Transport Layer Fits



Basic Transport Questions

BASIC QUESTIONS

1) What function does a transport provide?
2) What is a connection?
3) Why not have just one connection for all data?
4) Why not keep connections up always?
5) How do we address the receiving process in the receiver machine?



P1,..P4 denote processes (mail, ftp)

--- lines denote connections

Basic Transport Answers

BASIC QUESTIONS

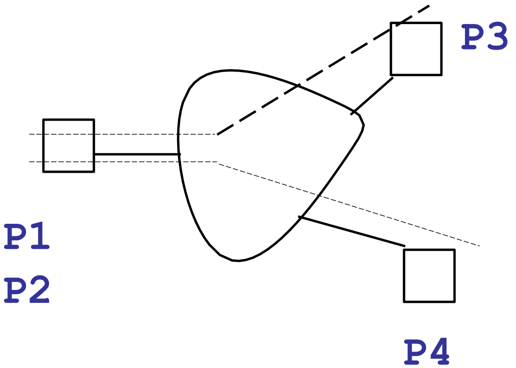
1) Reliable or Unreliable data Delivery between processes

2) Shared state for each process pair that enables delivery

3) Fast processes would be hostage to slow processes

4) Too many possible process pairs→ close + set up connections

5) Need an OS independent mechanism: ports



P1,..P4 denote processes (mail, ftp)

--- lines denote connections

Transport Layer Tasks

- Multiplexing (UDP does only this, so does TCP)
- Reliability (TCP only)
- Flow Control (TCP only)
- Congestion Control (TCP only)

Naming Processes/Services

- Process here is an abstract term for your Web browser (HTTP), Email servers (SMTP), hostname translation (DNS)

- How do we identify for remote communication?
  - Process id or memory address are OS-specific and transient

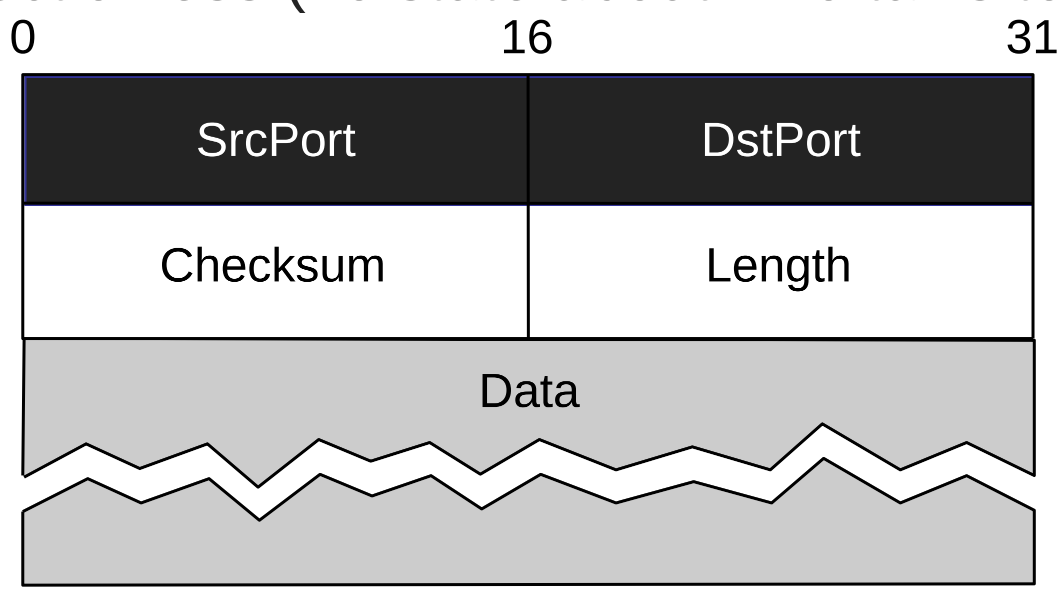
- So TCP and UDP use **ports**
  - 16-bit integers representing mailboxes that processes “rent”
  - Identify process uniquely as (IP address, protocol, port)

Picking Port Numbers

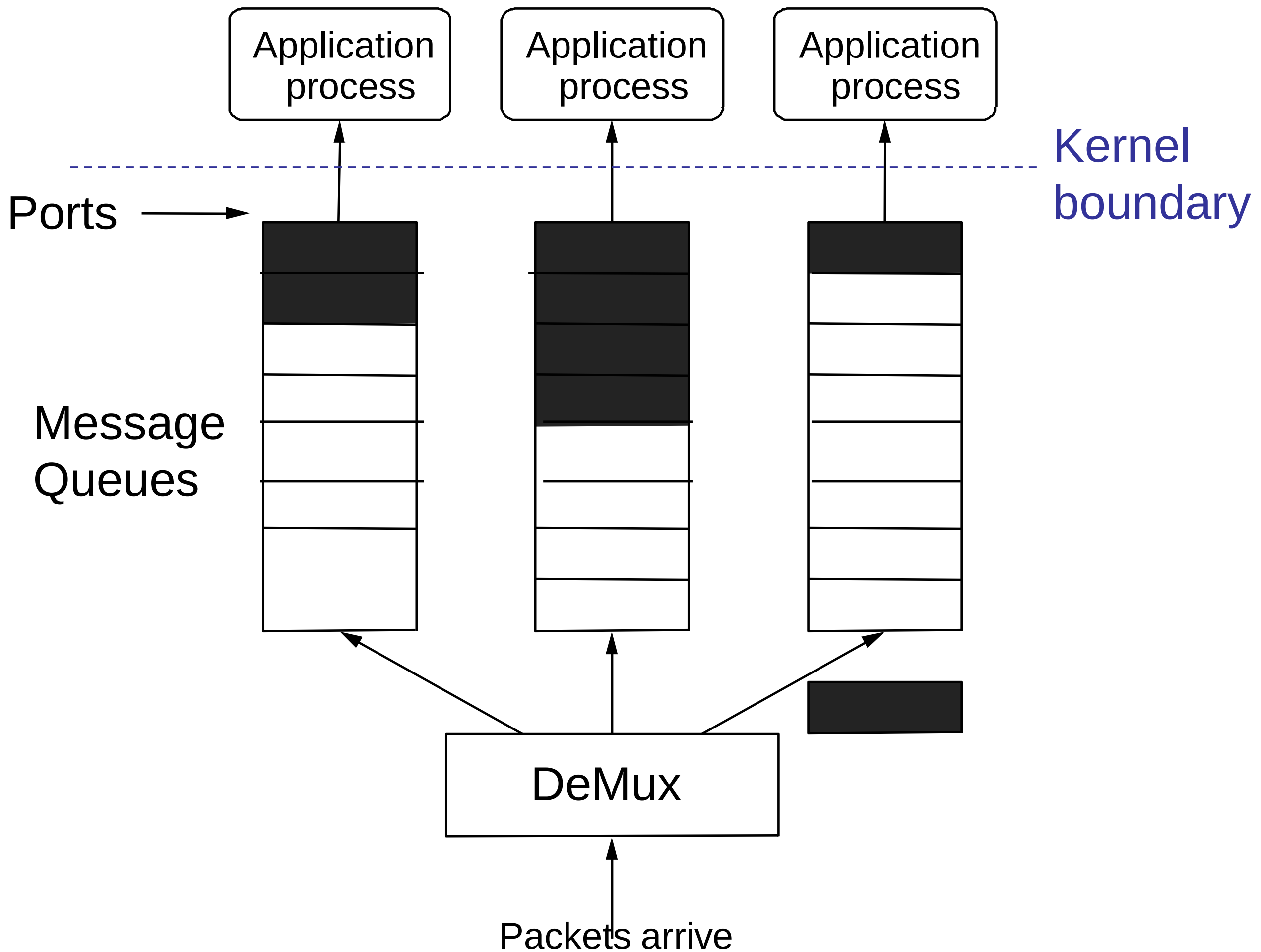
- We still have the problem of allocating port numbers
  - What port should a Web server use on host *X*?
  - To what port should you send to contact that Web server?
- Servers typically bind to well-known port numbers
  - e.g., HTTP 80, SMTP 25, DNS 53, … look in /etc/services
  - Ports below 1024 traditionally reserved for well-known services
- Clients use OS-assigned temporary (ephemeral) ports
  - Above 1024, recycled by OS when client finished

User Datagram Protocol (UDP)

- Provides *unreliable message delivery* between processes. So what does it do? Multiplexing!
  - Source port filled in by OS as message is sent
  - Destination port identifies UDP delivery queue at endpoint
- Connectionless (no state about who talks to whom)

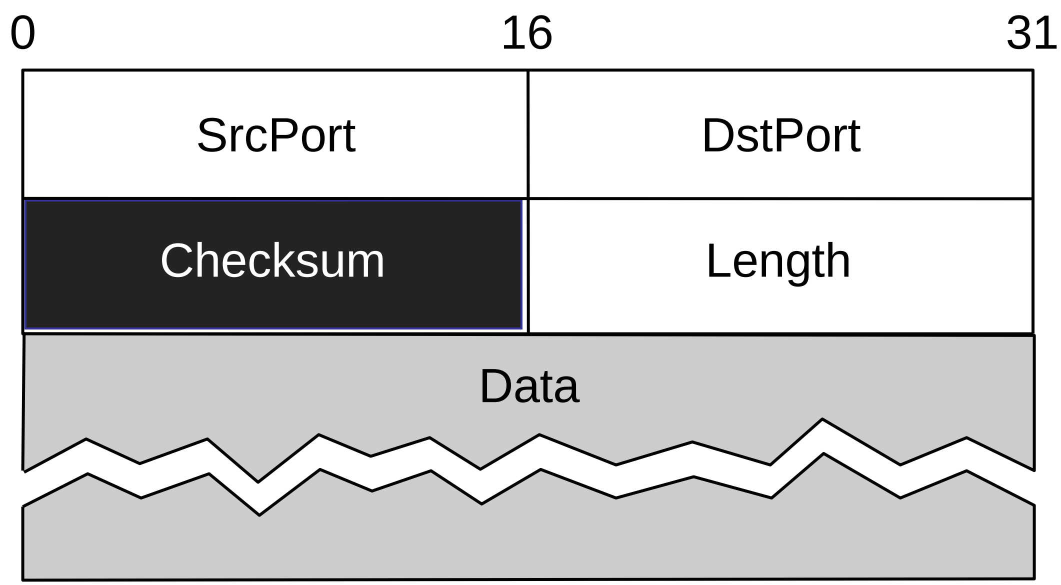


UDP Delivery



UDP Checksum

- UDP includes optional protection against errors
  - Checksum intended as an end-to-end check on delivery
  - So it covers data, UDP header, and IP pseudoheader



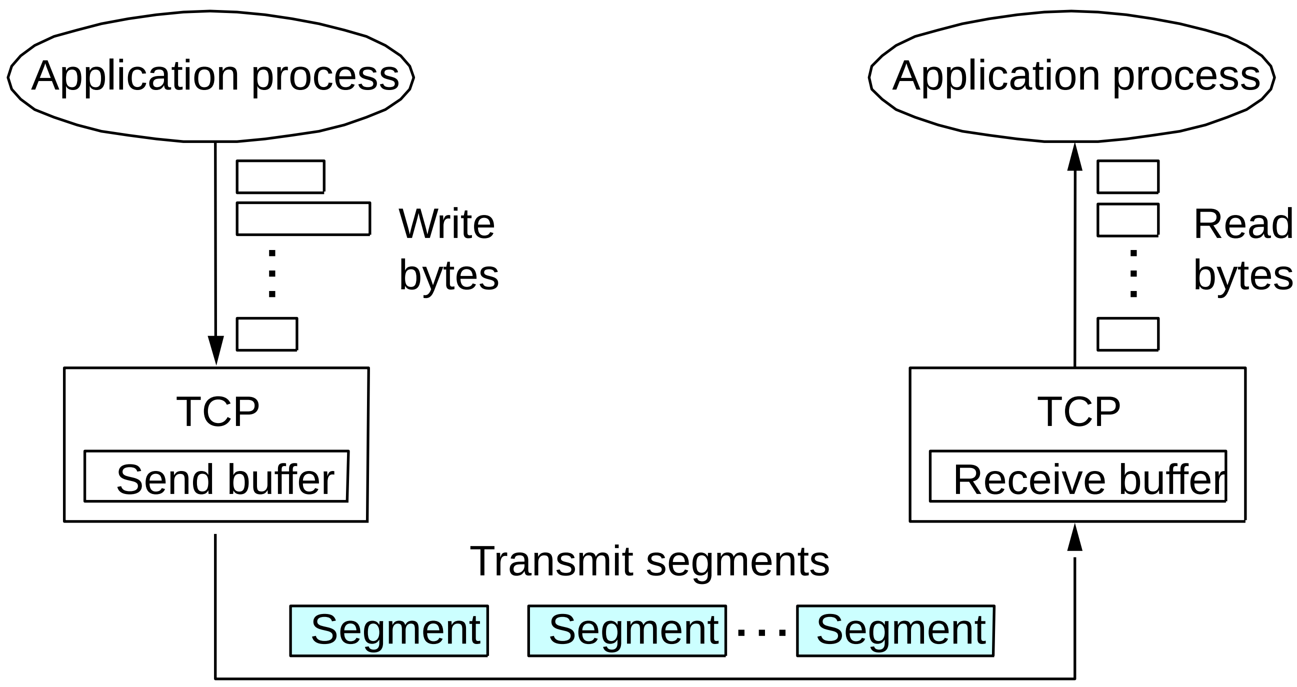
Applications for UDP

- Streaming media (e.g., live video)
- DNS (Domain Name Service)
- NTP (Network Time Protocol) (synchronizing clocks)
- FPS multi-player video games (e.g., Call of Duty)
- Why might UDP be appropriate for these?

Transmission Control Protocol

- Reliable bi-directional bytestream between processes
  - Uses a sliding window protocol for efficient transfer
- Connection-oriented
  - Conversation between two endpoints with beginning and end
- Flow control (generalization of sliding window)
  - Prevents sender from over-running receiver buffers
  - (tell sender how much buffer is left at receiver)
- Congestion control (next lecture)
  - Prevents sender from over-running network capacity

TCP Delivery



TCP like reliable data link

- Remember we said that when we did reliable data links that TCP would be similar (but end-to-end)
- This is where we “cash in” for all the hard work we did in sliding windows, go back N, Restart Protocols etc
- As a first approximation, TCP takes the bytes the user writes to the queue, packages them in segments, adds a sequence number and does Go Back N
- But there are differences we need to understand

Differences between Data Link reliability and TCP – Part 1

- Network instead of single FIFO link **We"ll do this second**
  - packets can be delayed for large amounts of time
  - duplicates can be created by packet looping: delayed duplicates imply need for large sequence numbers.
  - packets can be reordered by route changes.
- Connection management **We"ll do this first**
  - Only done for Data Link when a link crashes or comes up
  - Lots of clients dynamically requesting connections
  - HDLC didn't work: here more at stake, have to do it right.

Differences between Data Link reliability and TCP – Part 2

- Data link only needs speed matching between receiver and sender (flow control). Here we also need speed matching between sender and network (congestion control)
- Transport needs to dynamically round-trip delay to set retransmit timers.

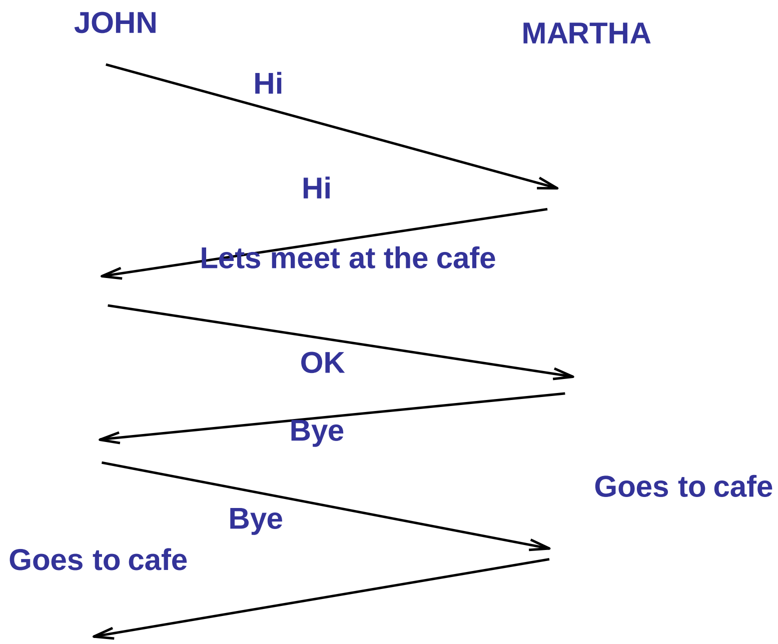
We"ll do these next lecture

Part 1:Connection Set up

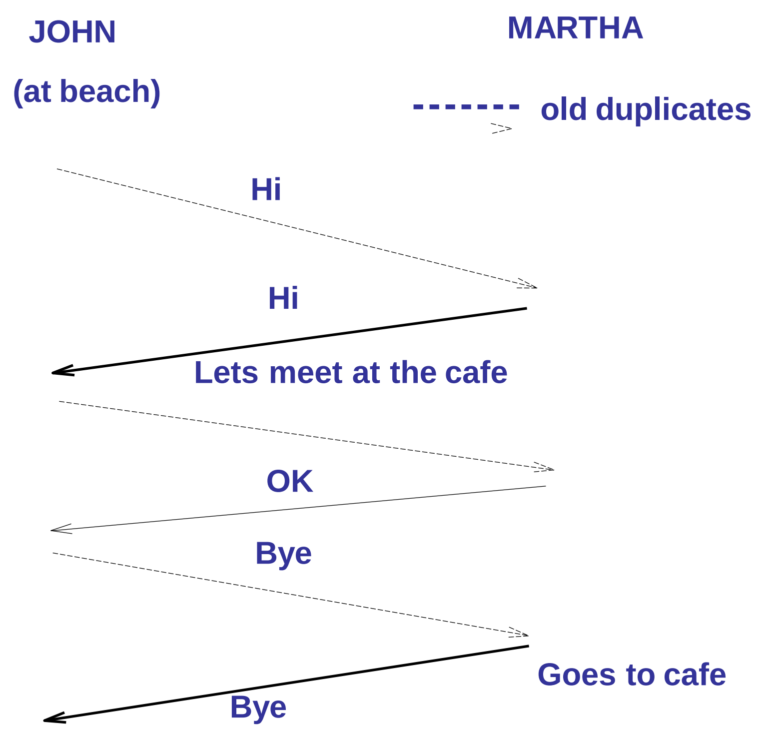
- Both sender and receiver must be ready before we start to transfer the data
  - Sender and receiver need to agree on a set of parameters
  - Most important: sequence number space in each direction
  - Lots of other parameters: e.g., the Maximum Segment Size

- Handshake protocols: setup state between two oblivious endpoints
  - Need to deal with delayed and reordered packets
  - Lets illustrate the problem with the sad tale of John and Martha

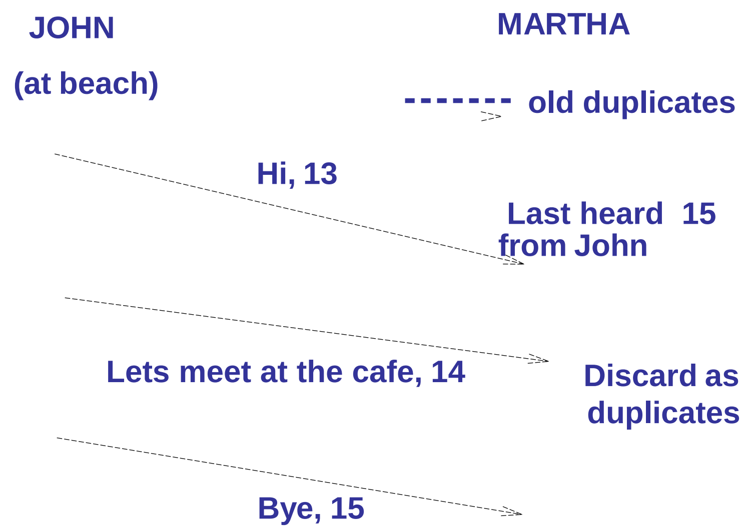
Going to the Cafe Protocol



Misery caused by duplicates

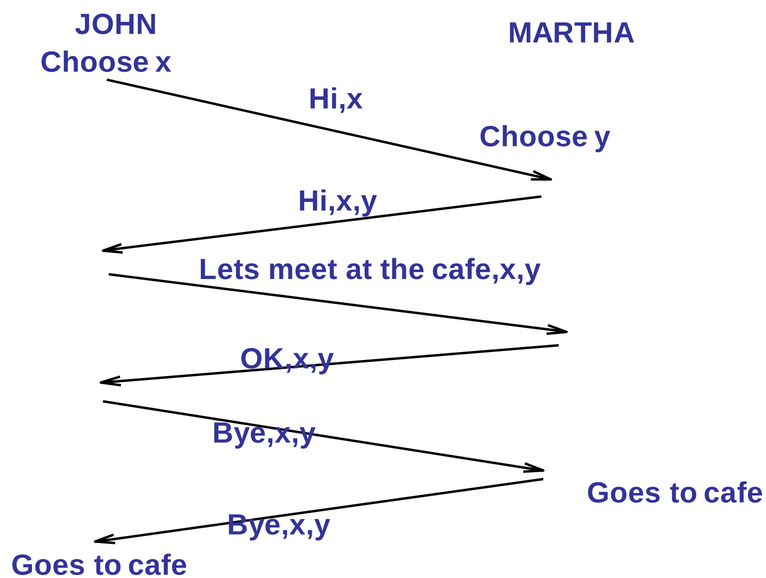


One way out: lots of state



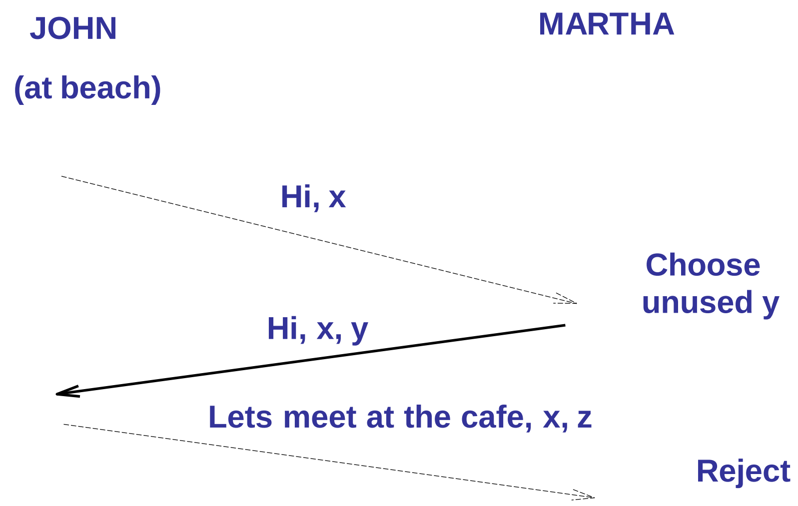
Martha has to remember last number from John
for worst−case packet lifetime. Called
Timer−based connection management.

TCP’s way: 3-way handshake

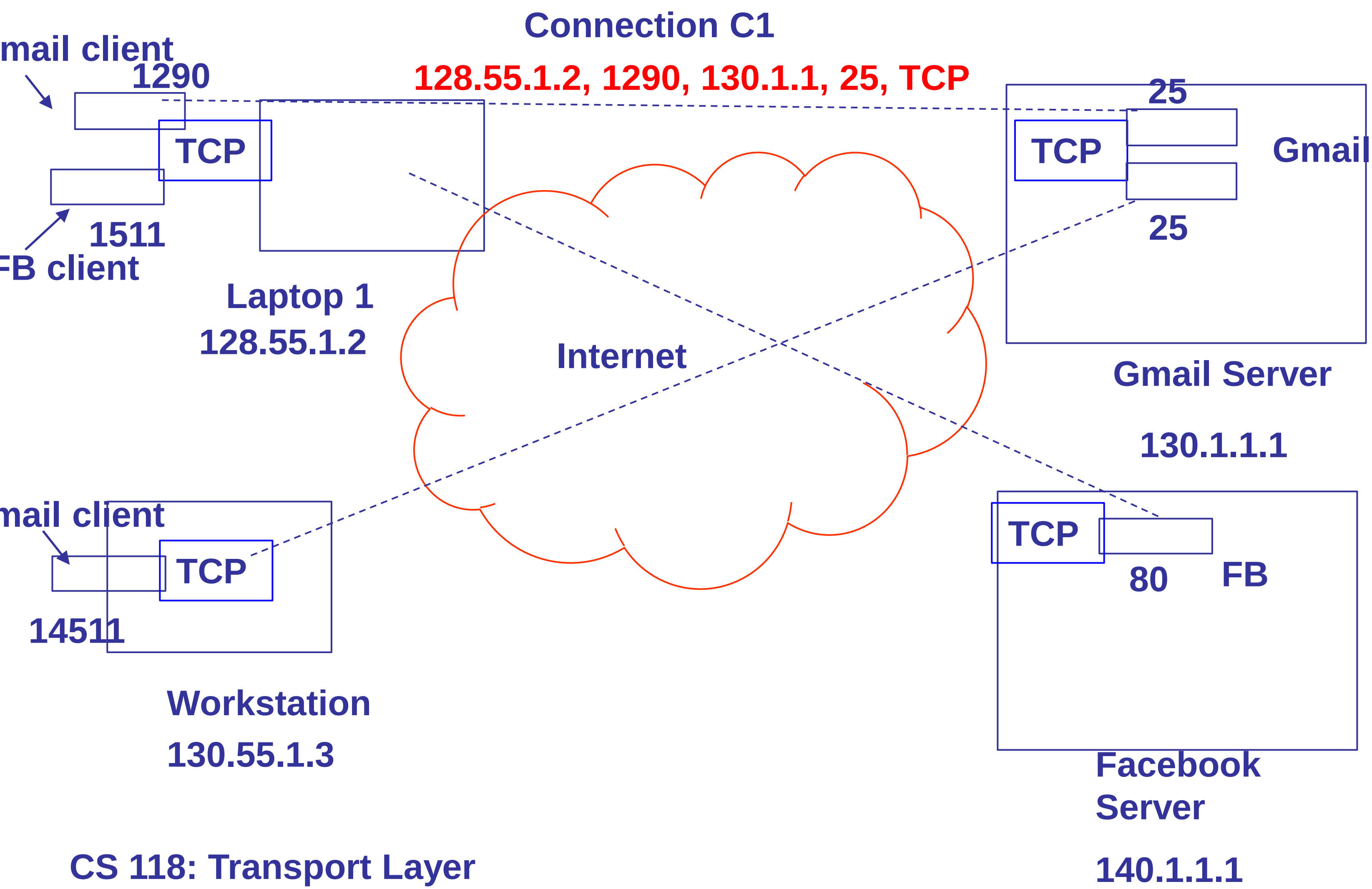


(x is validation number unused before by x)

Why nonces defend against delayed duplicates

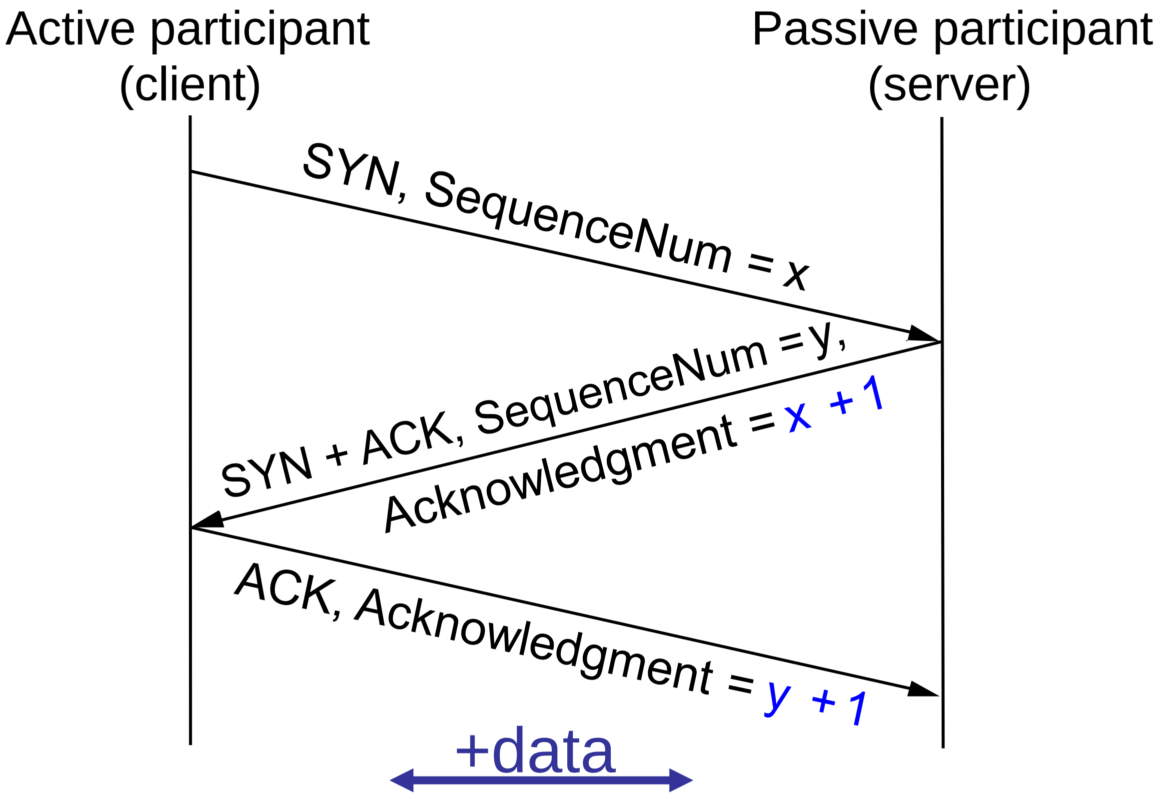


Connection Names: 4-tuples



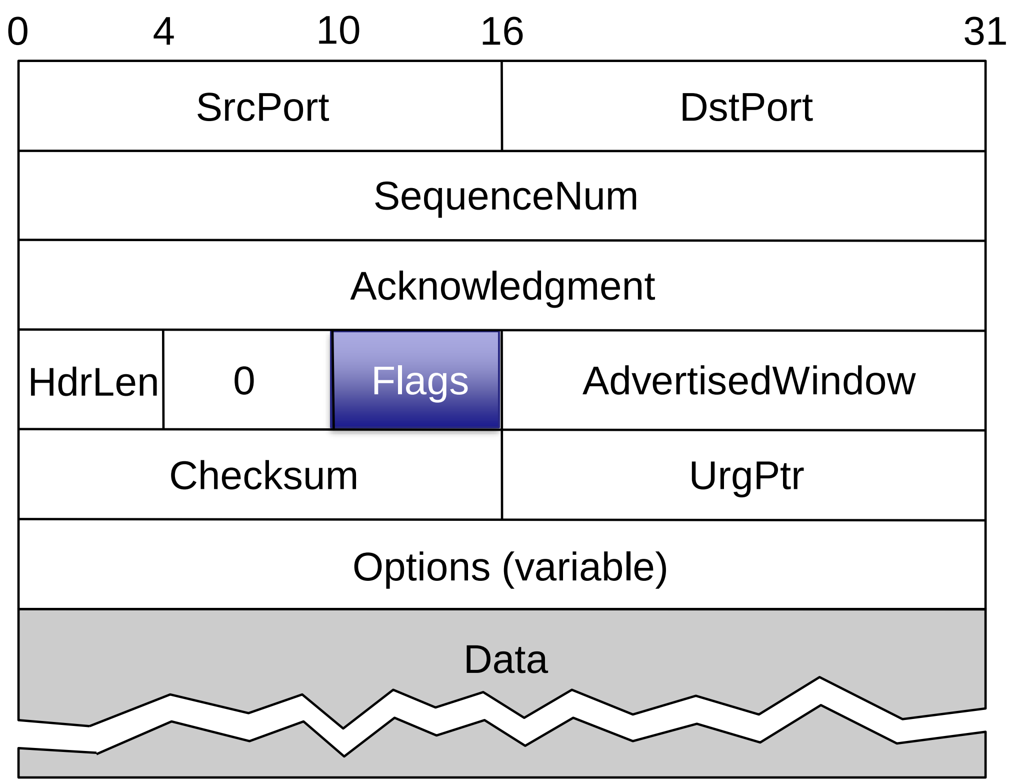
Three-Way Handshake in TCP

- Opens both directions for transfer



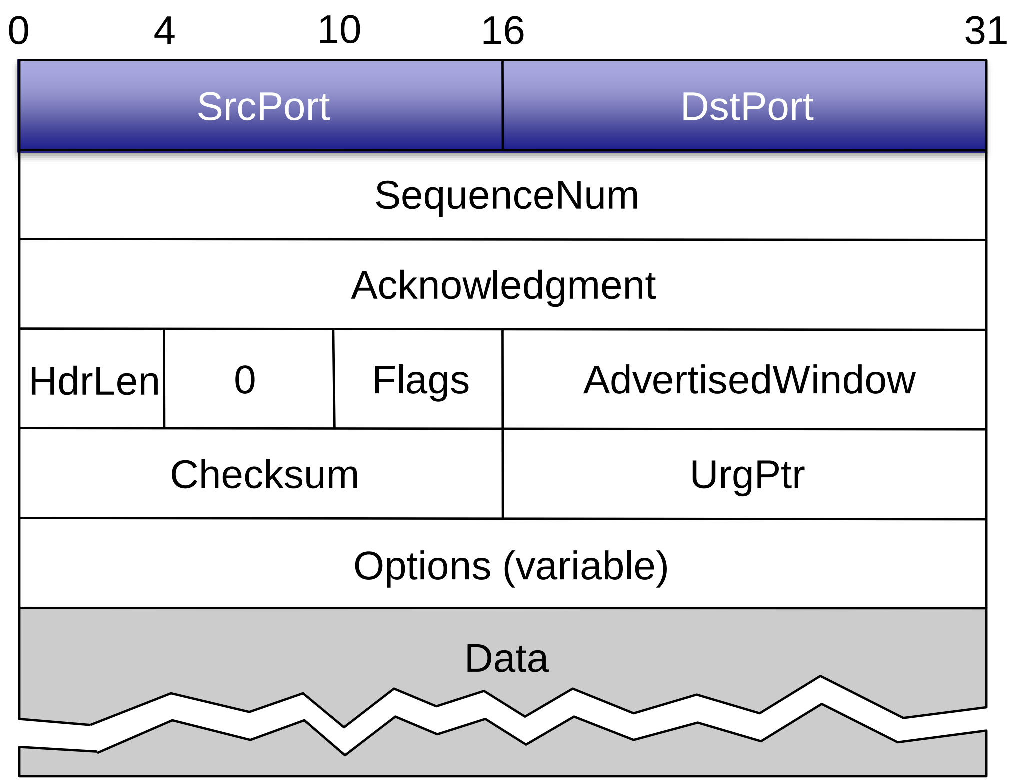
TCP Header Format

- Flags may be ACK, SYN, FIN, URG, PSH, RST



TCP Header Format

- Ports plus IP addresses identify a connection (4-tuple)



3-way handshake details

- We could abbreviate this setup, but it was chosen to be robust, especially against delayed duplicates
  - Three-way handshake first described inTomlinson 1975
- Choice of changing initial sequence numbers (ISNs) minimizes the chance of hosts that crash getting confused by a previous incarnation of a connection
- How to choose ISNs?
  - Maximize period between reuse
  - Minimize ability to guess (why?)

3-way handshake in TCP

- Server: If in LISTEN and SYN arrives, then transition to SYN_RCVD state, replying with ACK+SYN.
- Client: active open, send SYN segment and transition to SYN_SENT.
- Arrival of SYN+ACK causes the client to move to ESTABLISHED and send an ack
- When this ACK arrives the server finally moves to the ESTABLISHED state.

So now how do we disconnect

1) Need timers anyway to get rid of connection state to dead nodes.

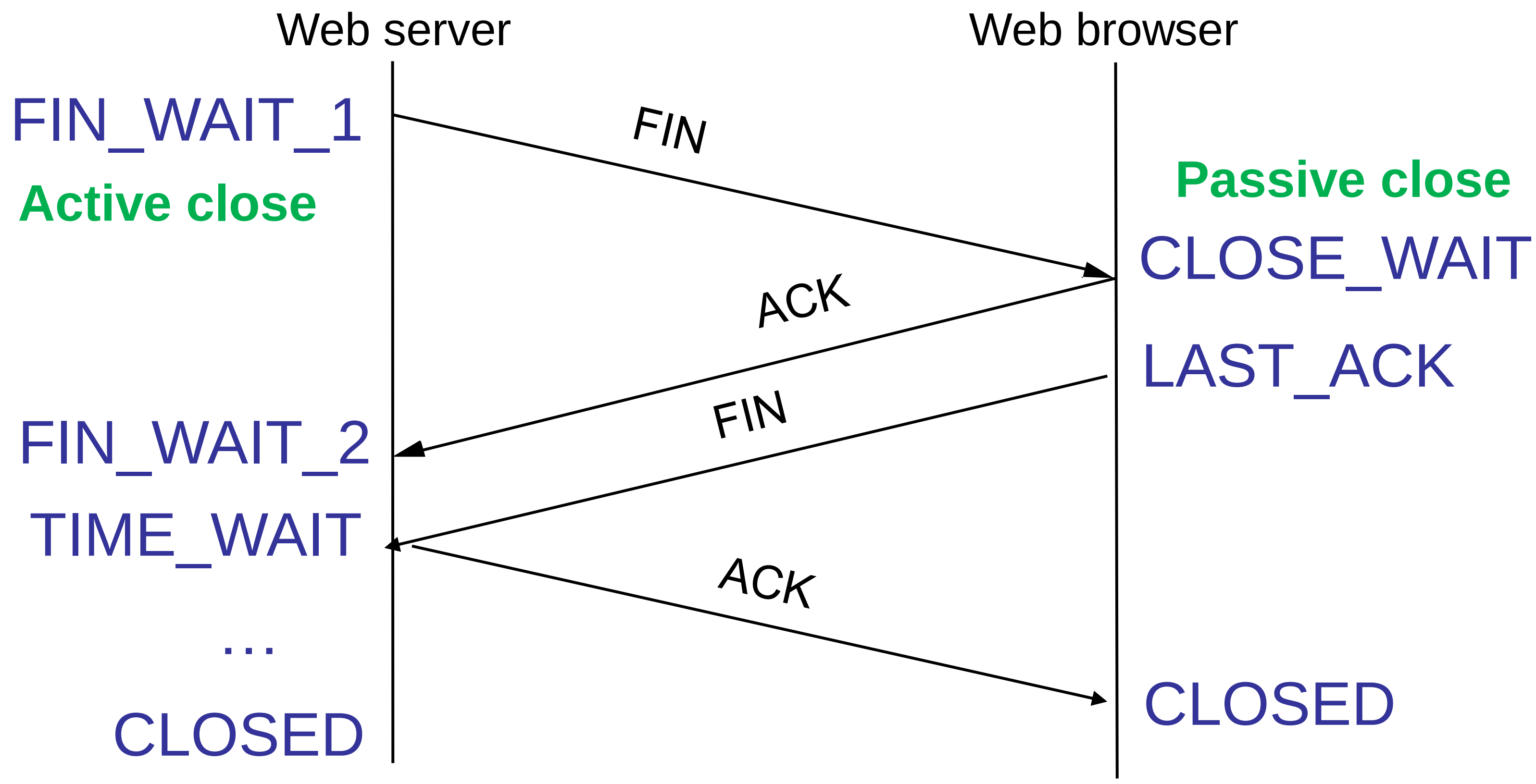
2) However, timer should be large so that "keepalive" hello overhead is low.

3) If communication is working, would prefer graceful closing (so receiver process knows quickly) to long timers.

4) Hence 3 phase disconnect handshake

After sending disconnect and receiving disconnect ack, both sender and receiver set short timers.

TCP Connection Teardown



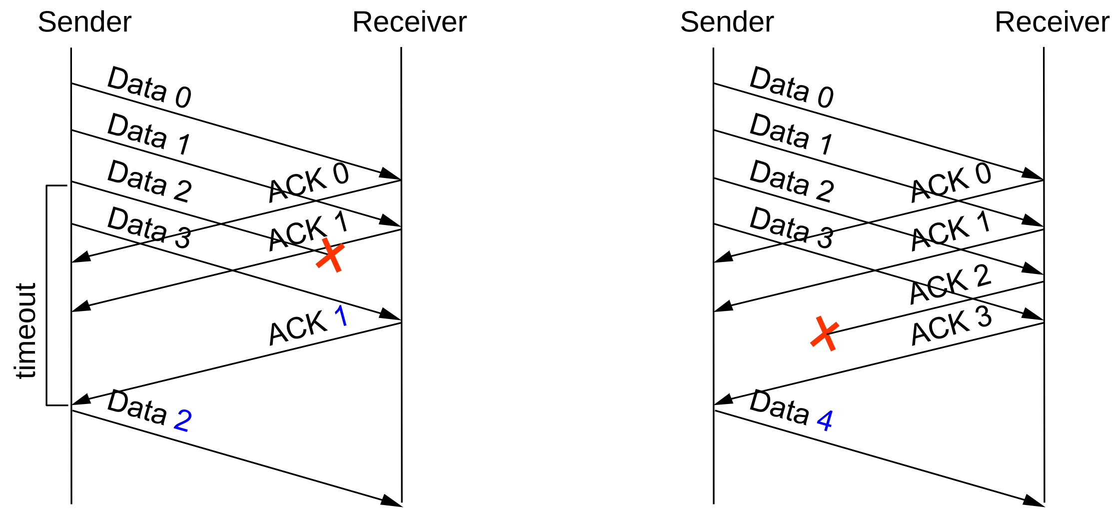
The TIME_WAIT State

- We wait 2*MSL (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
- Real life: Abortive close
  - Don't wait for 2*MSL, simply send Reset packet (RST)

Part 2: Reliable delivery

- Usual sequence numbers except:
  - Very large to deal with out of order (modulus > 2 W etc. only works on FIFO links)
  - TCP numbers bytes not segments: allows it to change packet size in the middle of a connection
  - The sequence numbers don't start with 0 but with an ISN.
- Reliable Mechanisms similar except:
  - TCP has a quicker way to react to lost messages
  - TCP does a crude form of selective reject not go-back N
  - TCP does flow control by allowing a dynamic window which receiver can set to reduce traffic rate (next lecture)

Remember Go-Back-*N*

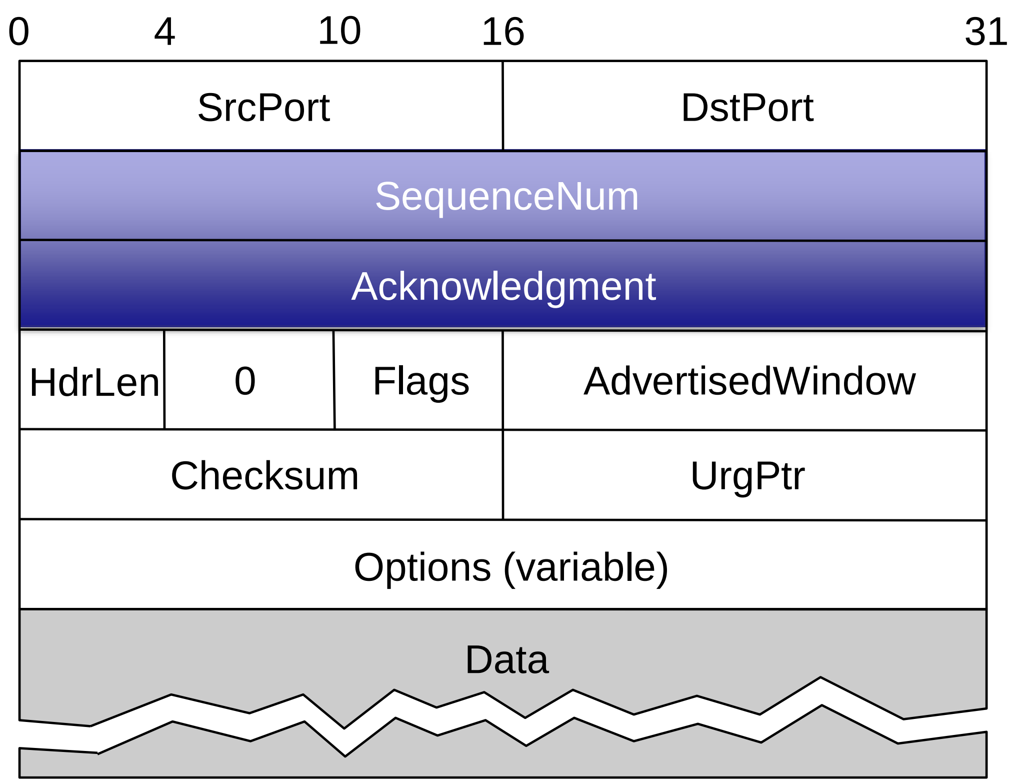


- Retransmit all packets from point of loss
  - Packets sent after loss event are ignored (i.e., sent again)

- Simple to implement (receiver very simple)
- Sender controls how much data is "in flight"

TCP Header Format

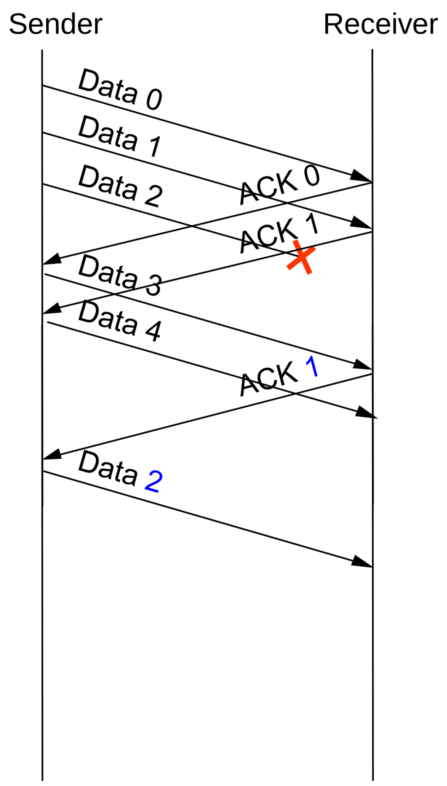
- Sequence, Ack numbers used for the sliding window
  - How big a window? Flow control/congestion control determine



Deciding When to Retransmit

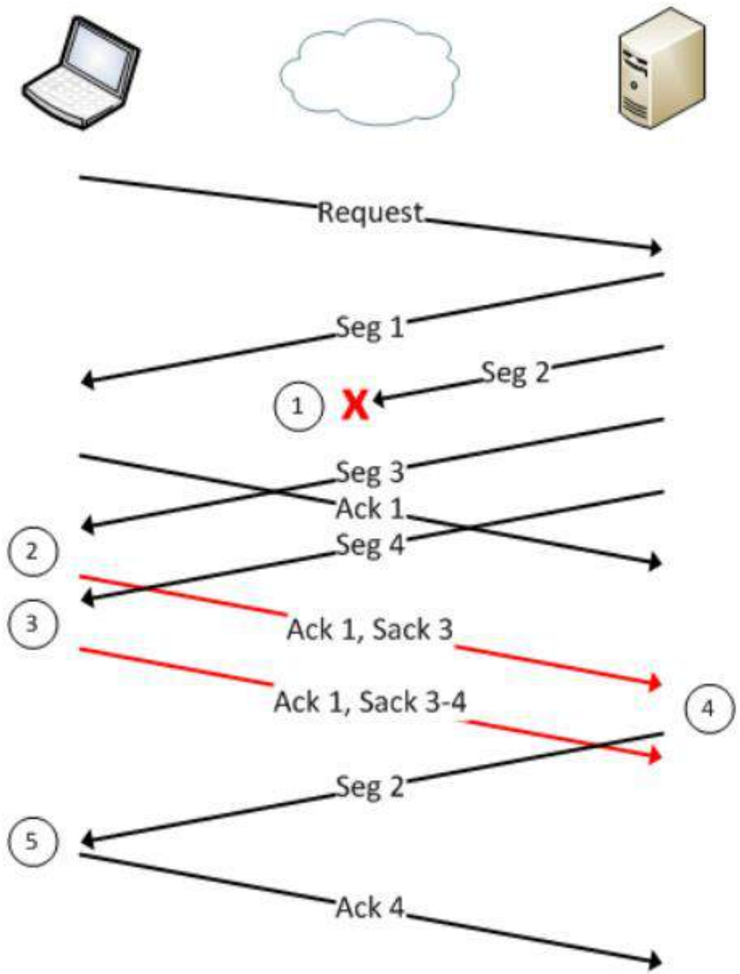
- How do you know when a packet has been lost?
  - Ultimately sender uses timers to decide when to retransmit
- But how long should the timer be?
  - Too long: inefficient (large delays, poor use of bandwidth)
  - Too short: may retransmit unnecessarily (causing extra traffic)
- Right timer is based on the round-trip time (RTT)
  - Which can vary greatly so we need to measure (next lecture)
  - But OS timer granularity makes it large (msec)
  - So we need another trick for common case error recovery

TCP Trick: Fast retransmit



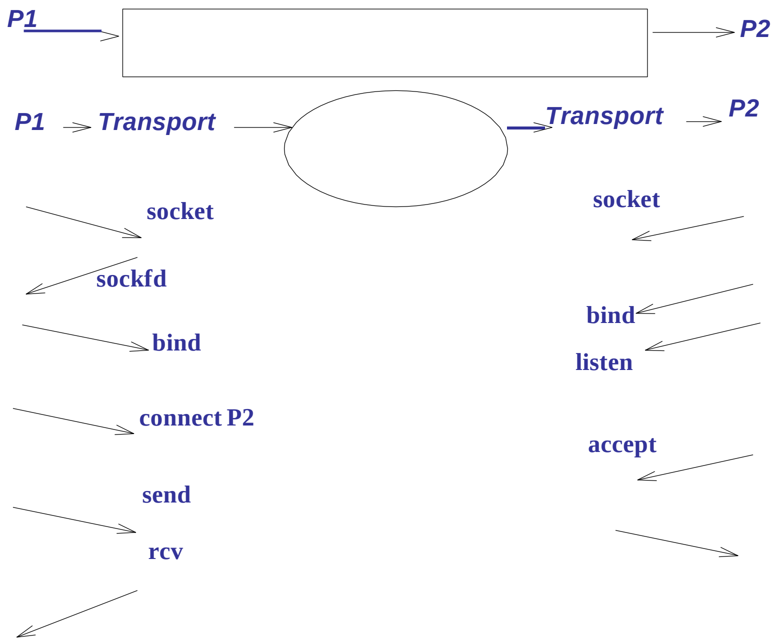
- Don't bother waiting
  - Receipt of duplicate acknowledgement (dupACK) indicates loss
  - Retransmit immediately
- Used in TCP
  - Need to be careful if frames can be reordered
  - Today's TCP identifies a loss if there are three duplicate ACKs in a row

Now playing: TCP SACK

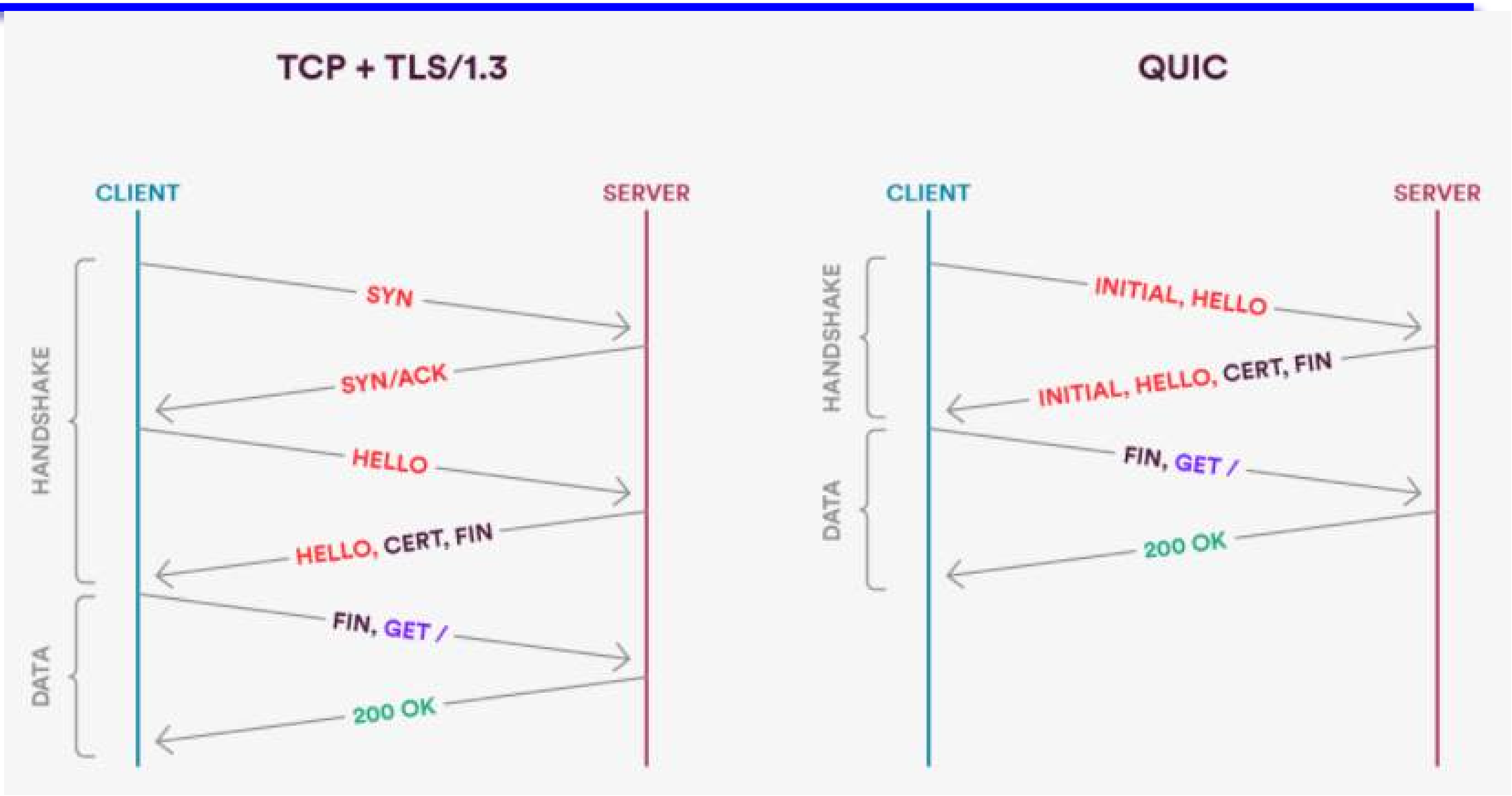


Has some limitations: like 3 SACK blocks

Socket Interface



QUIK first connection to server



QUIC: what's the latency trick

- Idea 1: Combine security handshake and sequence number handshake on first connection to server
- Idea 2: If server remembers information about client, need 0 handshakes on later connections
- Three way handshakes are required because server forgets info on client. Important in old days but no longer as severs have massive memory
- A round trip is a big deal (several hundred msec across US) at today's high speeds

QUIC: what else is new

- Stream multiplexing: multiple streams in a single QUIC connection between client and server for HTTP/2
- No head of line blocking: can do HTTP/2 over a single TCP connection but a single loss stalls all streams. Not so in QUIC
- Shared congestion information: as we will see it takes TCP a long time to ramp up. In QUIC all congestion information is shared.
- Wave of future: 4% of all websites use QUIC (3/2020)