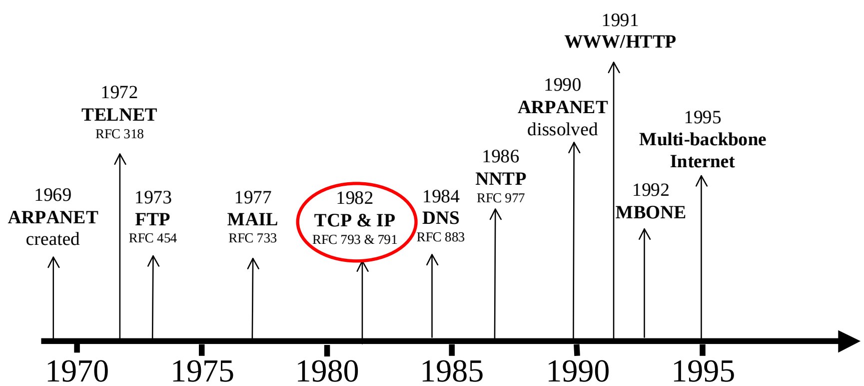
CSE/EE 461 Lecture 12
TCP

Tom Anderson

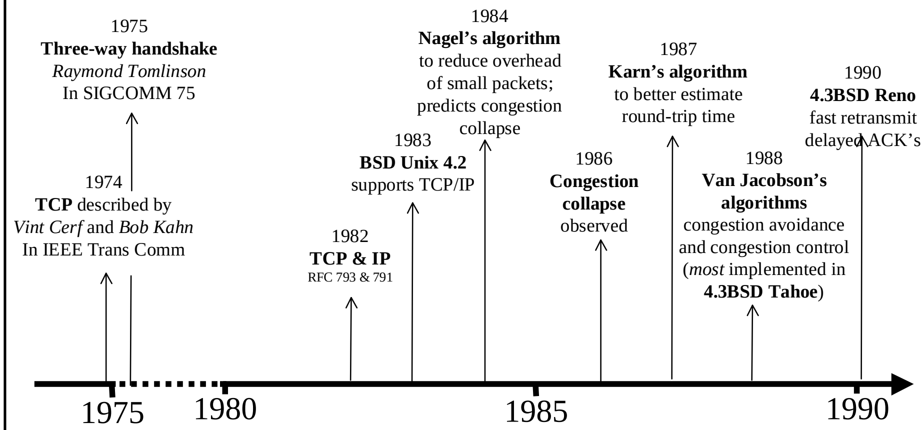
tom@cs.washington.edu

Peterson, Chapter 5.2, 6

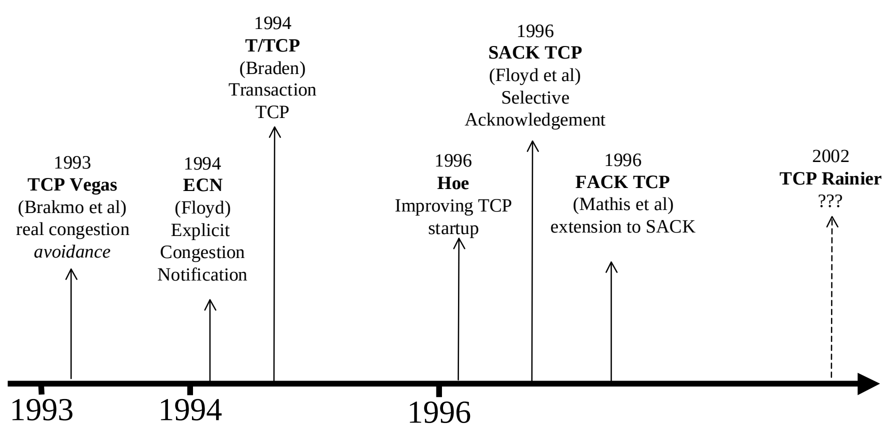
A brief Internet history...



TCP: This is your life...



TCP: After 1990

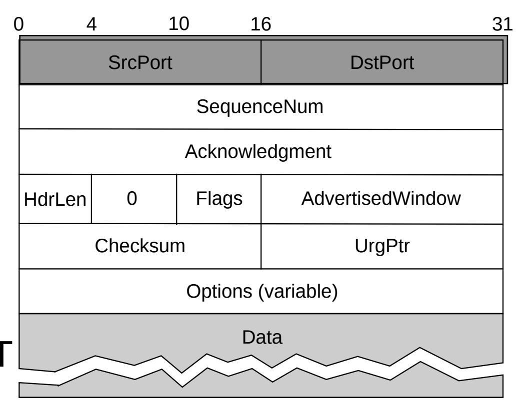


Transmission Control Protocol (TCP)

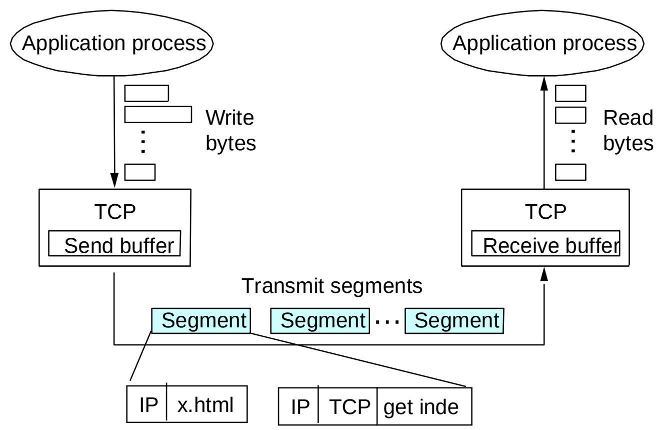
- Reliable bi-directional byte stream
  - No message boundaries
  - Ports as application endpoints
- Sliding window, go back N, RTT est, …
  - Highly tuned congestion control algorithm
- Connection setup
  - negotiate buffer sizes and initial seq #s
- Flow control
  - prevent sender from overrunning receiver buffers

TCP Packet Header

- Source, destination ports
- Sequence # (bytes being sent)
- Ack # (next byte expected)
- Receive window size
- Checksum
- Flags: SYN, FIN, RST



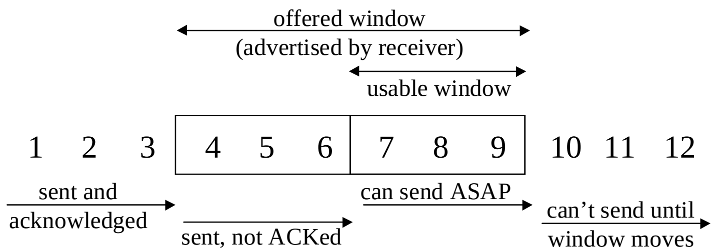
TCP Delivery



TCP Sliding Window

- Per-byte, not per-packet
  - send packet says "here are bytes j-k"
  - ack says "received up to byte k"
- Send buffer >= send window
  - can buffer writes in kernel before sending
  - writer blocks if try to write past send buffer
- Receive buffer >= receive window
  - buffer acked data in kernel, wait for reads
  - reader blocks if try to read past acked data

Visualizing the window

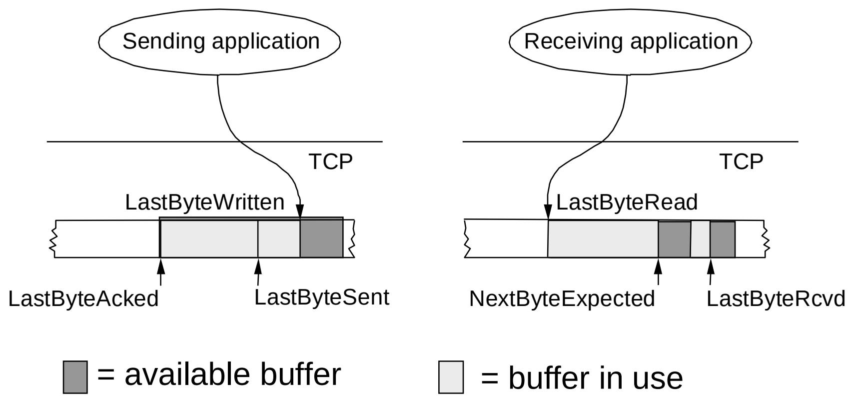


Left side of window advances when data is acknowledged.
Right side controlled by size of window advertisement

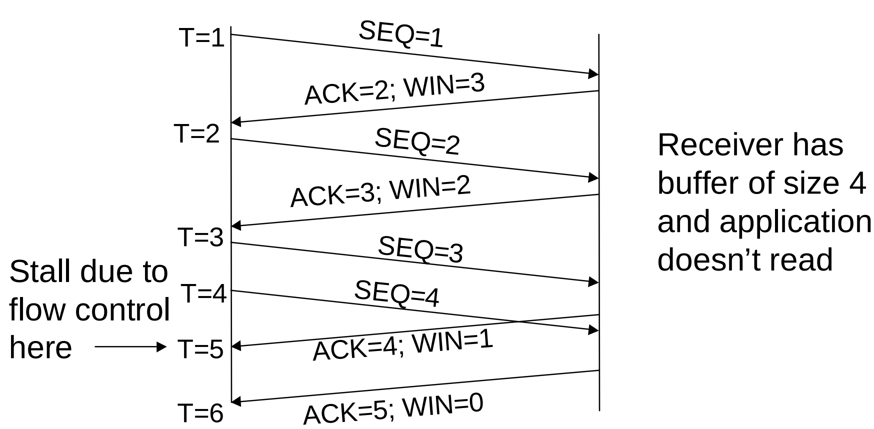
Flow Control

- What if sender process is faster than receiver process?
  - Data builds up in receive window
  - if data is acked, sender will send more!
  - If data is not acked, sender will retransmit!
- Sender must transmit data no faster than it can be consumed by the receiver
  - Receiver might be a slow machine
  - App might consume data slowly
- Sender sliding window <= free receiver buffer
  - Advertised window = # of free bytes; if zero, stop

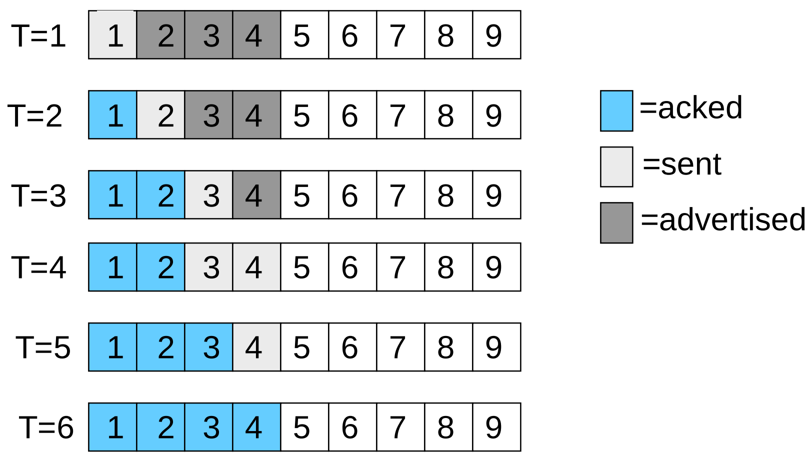
Sender and Receiver Buffering



Example – Exchange of Packets



Example – Buffer at Sender



How does sender know when to resume sending?

- If receive window = 0, sender stops
  - no data => no acks => no window updates
- Sender periodically pings receiver with one byte packet
  - receiver acks with current window size
- Why not have receiver ping sender?

Should sender be greedy (I)?

- Should sender transmit as soon as any space opens in receive window?
  - Silly window syndrome
    - receive window opens a few bytes
    - sender transmits little packet
    - receive window closes
- Solution (Clark, 1982): sender doesn’t resume sending until window is half open

Should sender be greedy (II)?

- App writes a few bytes; send a packet?
  - Don’t want to send a packet for every keystroke
  - If buffered writes >= max segment size
  - if app says “push” (ex: telnet, on carriage return)
  - after timeout (ex: 0.5 sec)
- Nagle’s algorithm
  - Never send two partial segments; wait for first to be acked, before sending next
  - Self-adaptive: can send lots of tinigrams if network is being responsive

Connections

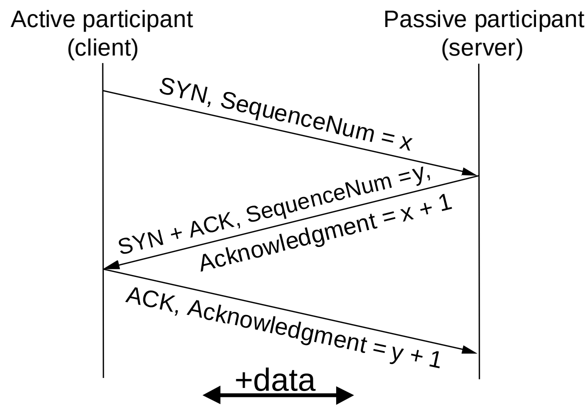
- Both sender and receiver must be ready before we start to transfer the data
  - Sender and receiver need to agree on a set of parameters
  - ex: receive buffer size, initial sliding window variables
- Sender and receiver must agree when transfer is over
  - Both sides must discard state
- This is signaling
  - It sets up/tears down state at the endpoints
  - Compare to "dialing" in the telephone network

TCP Connection Management

- Setup
  - assymetric 3-way handshake
- Transfer
  - sliding window; data and acks in both directions
- Teardown
  - symmetric 2-way handshake
- Client-server model
  - initiator (client) contacts server
  - listener (server) responds, provides service

Three-Way Handshake

- Opens both directions for transfer

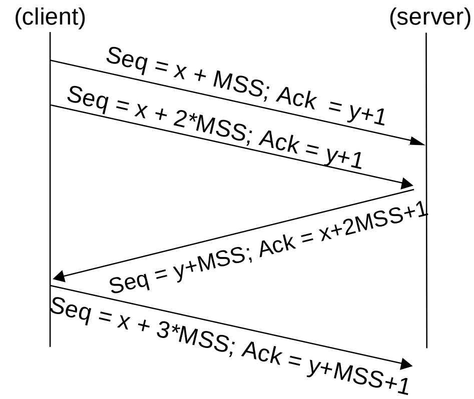


Do we need 3-way handshake?

- Allows both sides to
  - allocate state for buffer size, state variables, …
  - calculate estimated RTT, estimated MTU, etc.
- Helps prevent
  - Duplicates across incarnations
  - Intentional hijacking
    - random nonces => weak form of authentication
- Proposals to short-circuit
  - Persistent connections in HTTP (keep connection open)
  - Transactional TCP (save seq #, reuse on reopen)

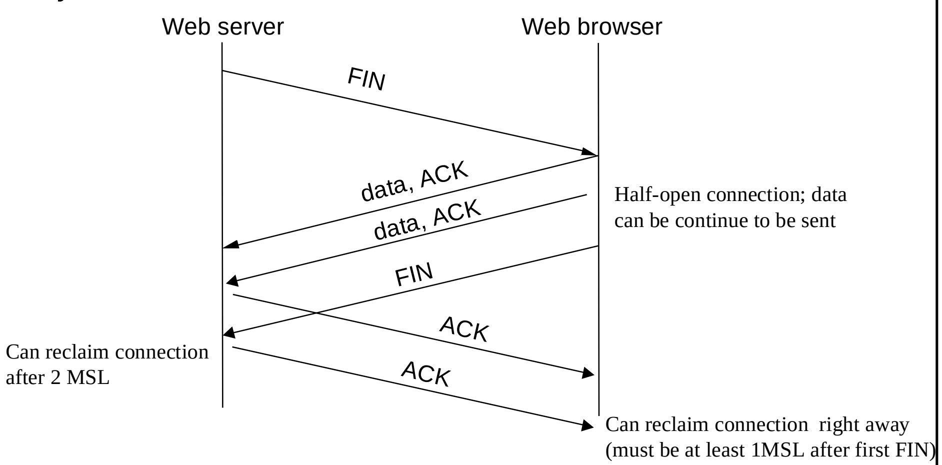
TCP Transfer

- Connection is bi-directional
  - acks can carry response data

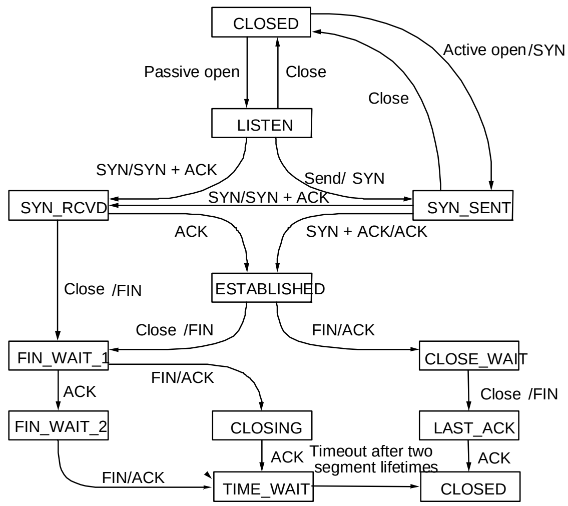


TCP Connection Teardown

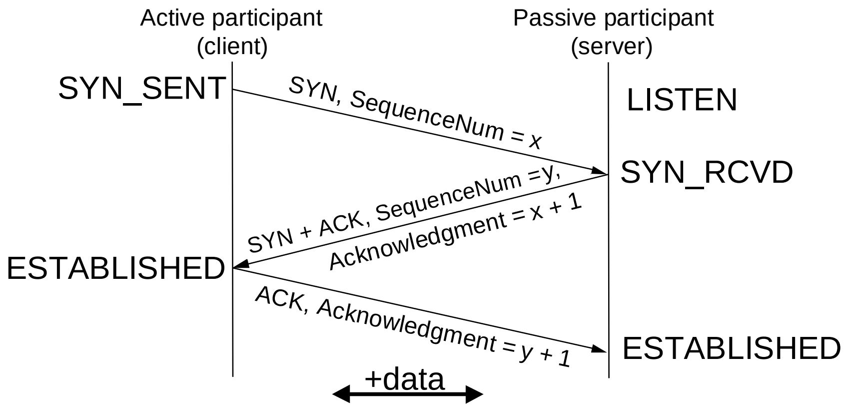
Symmetric: either side can close connection



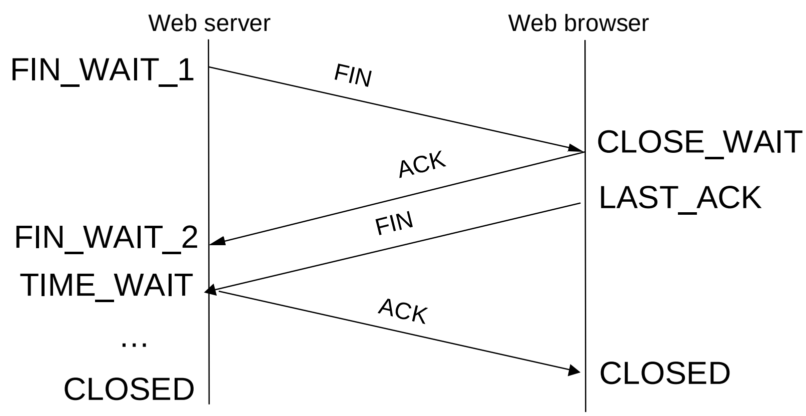
TCP State Transitions



TCP Connection Setup, with States



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