CMSC 417 Spring 2016 Lecture #14 3/28/2016

Agenda:

- ⇒ guest lecture by Eric Jeney
- ⇒ TCP
  - ☐ abstraction
  - ☐ header
  - ☐ state diagram
  - ☐ nagle's algorithm

Eric will cover 3/28/2016

UDP:
⇒ just another header on IP to allow delivery to applications vs. computers
⇒ "connectionless" = no state is maintained

TCP:
⇒ reliable
⇒ connection-oriented
⇒ byte stream

⇒ uses sliding windows for reliability, flow control and congestion control

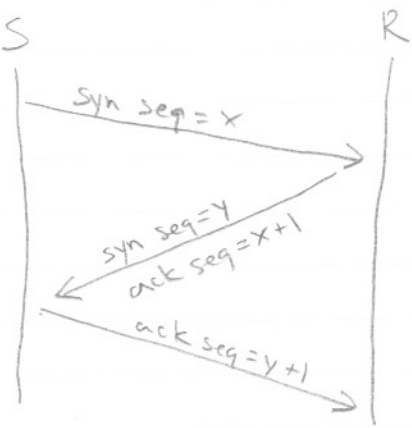
TCP Header

Flags:

SYN	FIN	RST	PSH	URG	ACK

⇒ synchonize
⇒ finish
⇒ reset
⇒ push
⇒ urgent
⇒ acknowledge

Setup:



⇒ all seq #s are in bytes
⇒ you ack the next byte you expect
⇒ SYN and FIN flags - consume one byte

Eric will cover 3/28/2016

TCP header

⇒ source port (16 bits)
⇒ dest port (16 bits)
⇒ seq # (32 bits)
⇒ ack # (32 bits)
⇒ data offset (4 bits) size of header in 32-bit words
⇒ reserved (6 bits) must be 0
⇒ flags (6 bits) see prev page
⇒ window size (16 bits) # of bytes receiver is willing to buffer
⇒ checksum (16 bits) over whole segment
⇒ urgent pointer (16 bits) offset from seq # indicating the last urgent byte
⇒ options (0-320 bits in 32-bit increments)

→ $(2^4-1) \times 32$ bits = 480 bits = 60 bytes max header

min size = 160 bits = 20 bytes

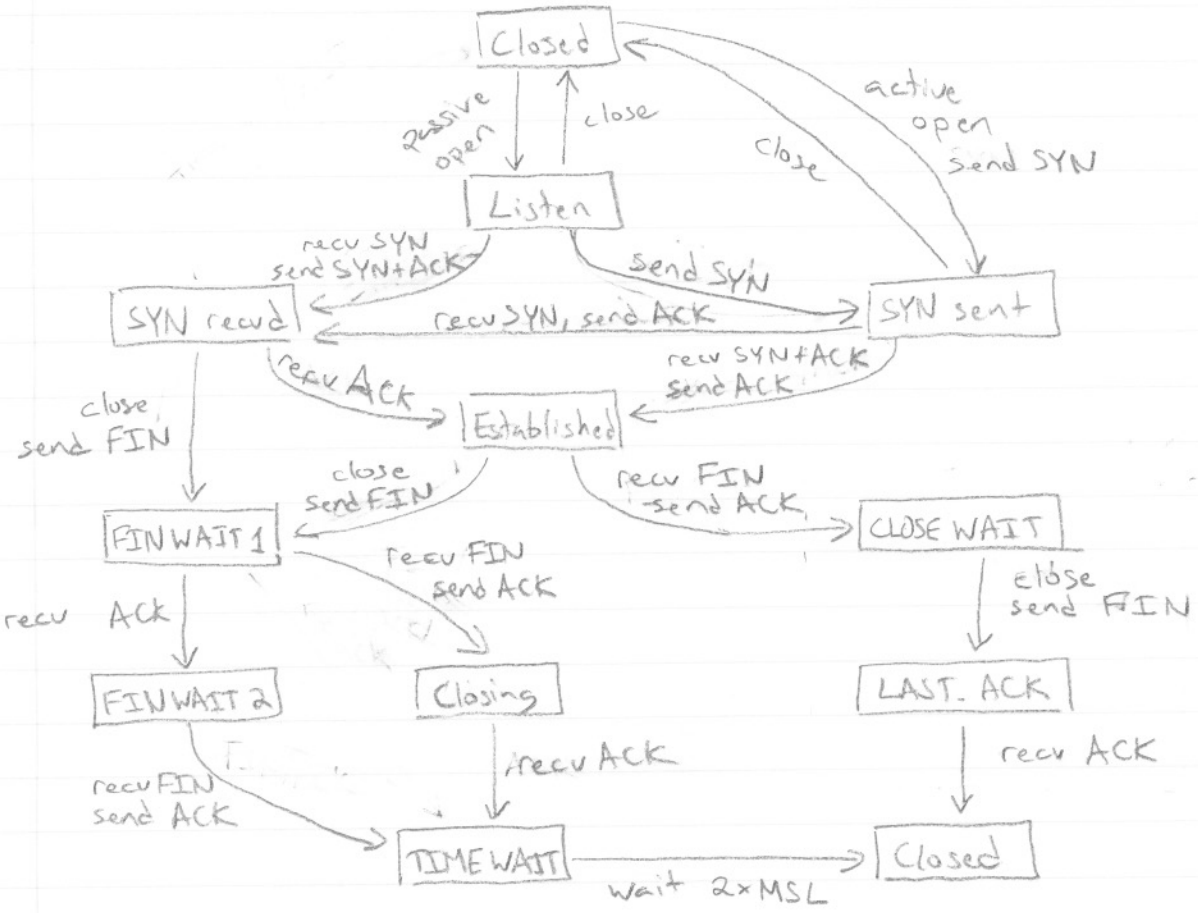
Flags

⇒ SYN, FIN and ACK will be clear shortly
⇒ RST = reset = something bad has happened, abandon the connection
⇒ PSH = push = this segment marks the end of a useful chunk of data, please deliver it to the app without waiting for more
⇒ URG = urgent = the data in this segment from the first byte to the urgent pointer is important (not really used, sometimes used in Telnet)

Eric will cover 3/28/2016

TCP State Diagram

From RFC 793



MSL = max segment lifetime

⇒ configurable

⇒ typically a few minutes

⇒ prevents segments from this connection that might still be out there from screwing up other connections

Notes:

⇒ one-side closed TCP connections

FIN_WAIT_1, FIN_WAIT_2, CLOSE_WAIT

⇒ TCP packets are called segments

Eric will cover 3/28/2016

Nagle's Alg

→ also true for apps that write small amounts of data w/ some negative consequences

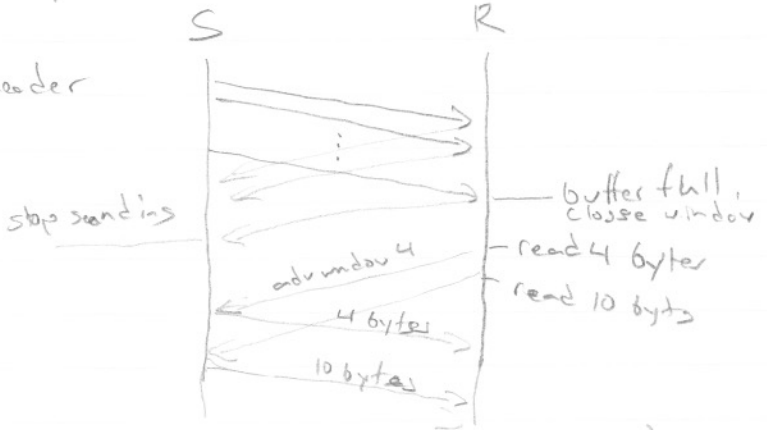
Problem

⇒ many apps will read relatively small amounts of data at a time, e.g., << MSS
- esp. when parsing binary data
- e.g., type-length-value fields
  - read type and length (often 4-8 bytes)
  - read length bytes (often small)

⇒ results in advertised window growing by small amounts

⇒ results in sender sending small amounts of data repeatedly — Silly Window Syndrome

⇒ wasteful: ≤ 10 bytes + 40 bytes min TCP+IP header
⇒ ≥ 400% overhead
- also lots more pkts means more pkt processing



solution: Nagle's Alg.

⇒ have at most one "tiny gram" (≤ MSS segment) in flight at a time — either
- wait for a full MSS of data & adv wnd, or
- wait for ack of last tiny gram

⇒ limits overhead

⇒ can be disabled — when would you do this? why?