I. Transmission Control Protocol (TCP)

Week 12

- RFC 793
- recall TCP is a reliable, connection based protocol
- **<u>services</u>** provided by TCP

1. virtual circuits

   - provide guaranteed connection
   - data exchange between VC is full duplex

2. application I/O management

   - internal addressing (port assignment)
   - connection setup/teardown
   - data transfer

3. network I/O management

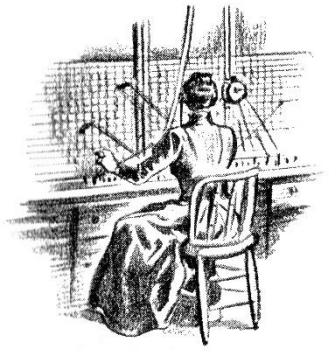
   - efficient segment sizing
   - MTU/MRU/buffers/header sizes

4. flow control

   - adjustment of send/receive rates

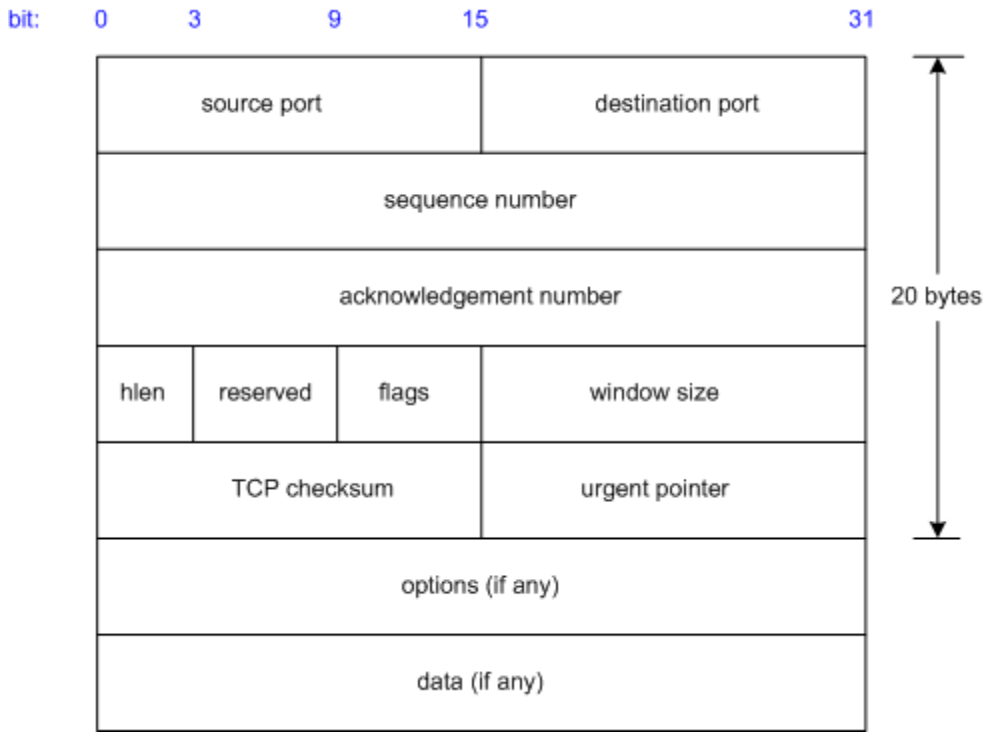
5. reliability

   - error detection
   - error correction



A Western Electric switchboard of the early '80's.

II. TCP PDU (see pg. 327)



- **source port** (16 bits): identifies points at which upper-layer source that created the data
- **destination port** (16 bits): identifies points at which upper-layer destination should receive the data
- **sequence** # (32 bits): identifies the first byte of data in the stream from the sender to the receiver
  - allows the destination to sort data in proper order
  - ISN: initial sequence number when VC is established
- **acknowledgement** # (32 bits): identifies the next sequence number the destination expects to receive
  - identifies all data up to, but not including, this number has been received
  - simplified, sequence # in + bytes data received = ack # out
- **header length** (4 bits): size of header in 32 bit multiples, only size of header, not the size of data (unlike UDP)

- **reserved** (6 bits): currently unused, set to zero
- **control flags** (6 bits): provide VC management services

    - URG (urgent)
    - ACK – every segment sent will set this (except for 1st one and reset)
    - PSH
    - RST
    - SYN – VC endpoints use to sync their sequence numbers (ISN's)
    - FIN

- **window**: flow control mechanism

- **checksum**: checksum of entire segment (header & data)

    - mandatory (unlike UDP), why?
    - if checksum bad, segment is dropped

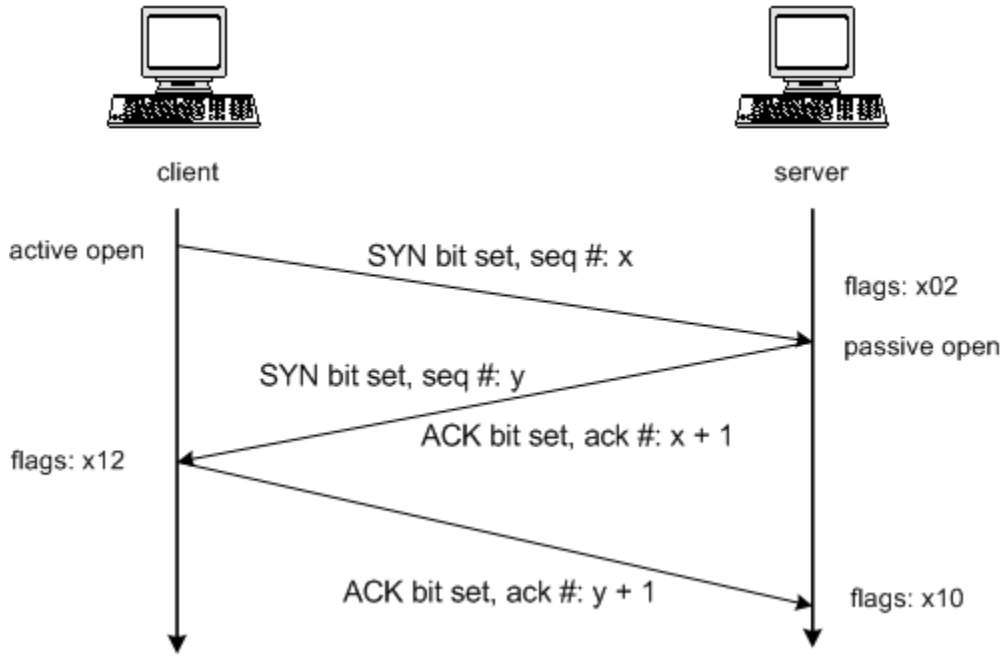
- **urgent**: indicates any urgent segments

- **options**: see pg 346

III. TCP Connection Establishment Sequence

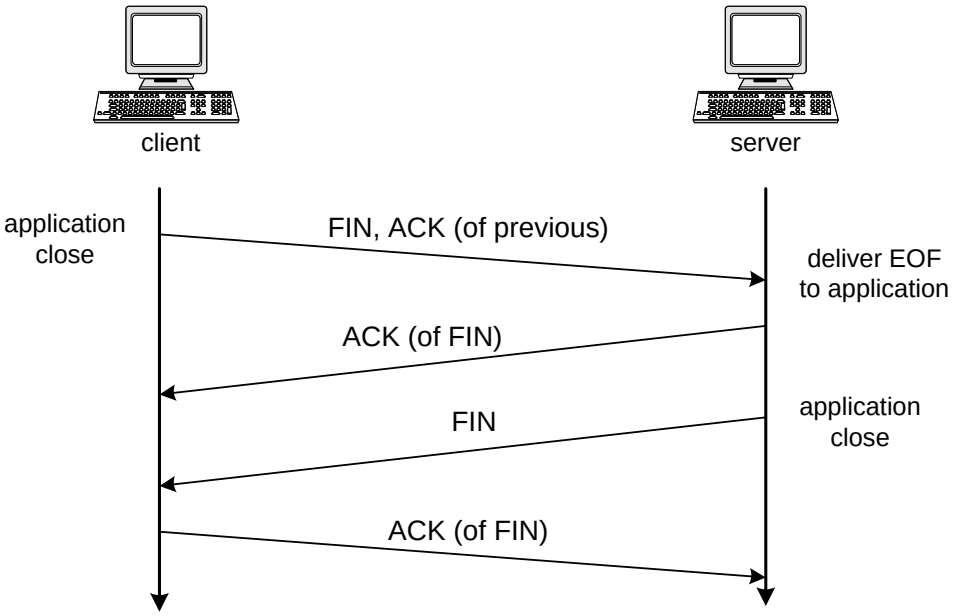
- referred to as the **3-way handshake**
- see pg 271 - 281
- recall ACKs set for every segment except the 1st (and RST)
- flag values:

32	16	8	4	2	1
URG	ACK	PSH	RST	SYN	FIN



IV. TCP Connection Teardown

- requires four segments to terminate a connection
- since TCP connection is full-duplex, each direction must be terminated independently
- when a side receives a FIN, it will send an ACK of the incoming sequence number + 1



Note 2nd and 3rd segments usually combined

from Stevens, pg 234

V. Incomplete Termination Types

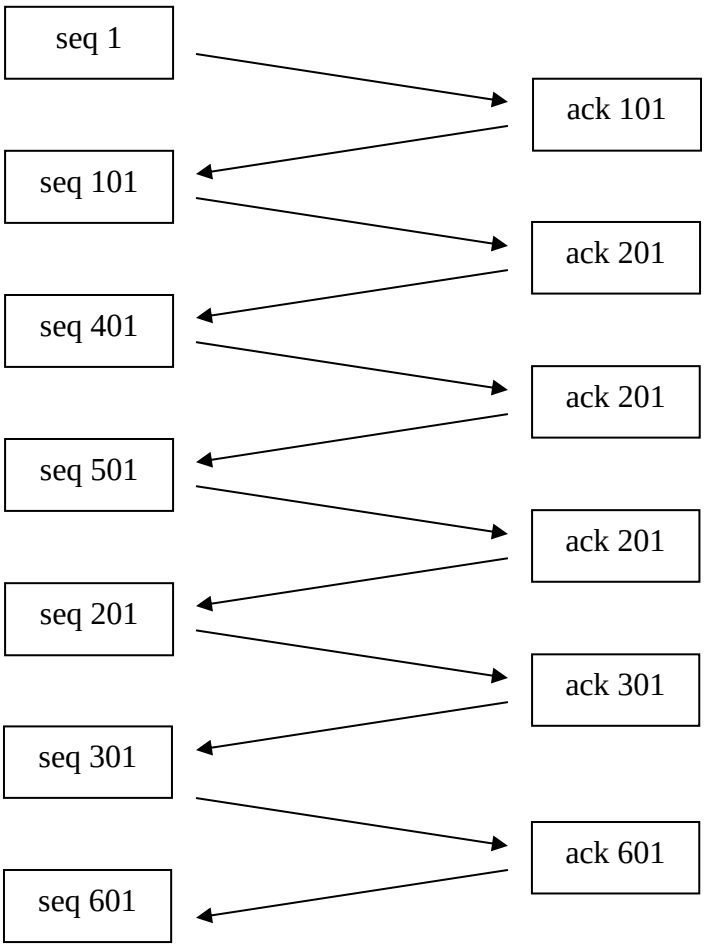
1. half close
   - one end of the VC sends a FIN, other continues sending data
   - see handout
2. half open
   - one end of the VC closes or aborts without knowledge of the other end
   - e.g. client turned off

VI. Reset

- sent when a TCP segment arrives which is not destined for a valid connection
- immediate connection termination on both sides of the VC, RST flag set
- what happens in UDP since no control flags? ICMP error, port unreachable

VII. Out of Order Arrival

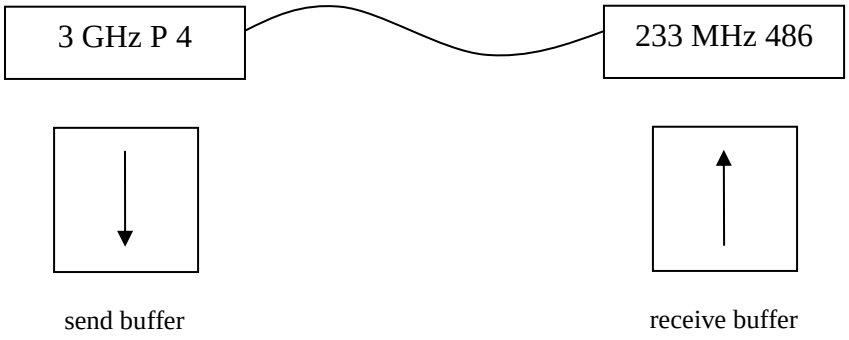
- since full-duplex, segments can arrive out of order
- segments can also get lost, see pg. 320



- out of order packets will be stored in buffer until all packets arrive and can be reassembled
- however, if missing segments do not show up in a timely manner, timers will expire and buffer will be emptied

VIII. TCP Flow Control

- recall TCP communicates full-duplex
- goal is to manage flow such that transmission is maximized and loss (overflow) is minimized
- both sender and receiver use buffers, must keep in buffer until that data is ACK'd
- but all hardware is not the same speed, e.g.



Simplest case - <u>Request/Reply:</u>

- **send** one segment, **wait** for an ACK
- slow, but can process/discard packet as soon as receiving an ACK
- not realistic, too much data flow

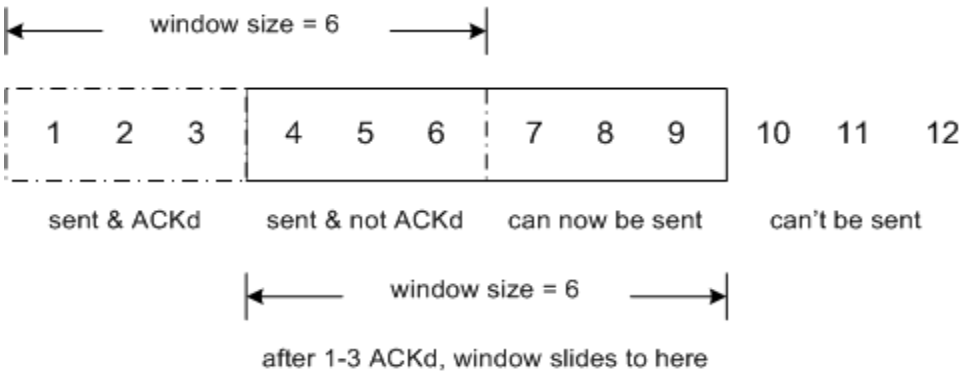
Receiver Based Flow Control

1. Receiver Window Size Adjustment (pg. 296, 379)

   - RFC 793
   - uses the window field returned in ACKs to tell a sender how much data the receiver can handle
   - window # specifies the # of octets the receiver is prepared to receive (i.e. that will fit into the receive buffer) before sending an ACK
   - see example by Hyojin Kim: http://media.pearsoncmg.com/aw/aw_kurose_network_2/applets/flow/flowcontrol.html

2. Sliding Receiver Windows (pg. 301)

   - RFC 793, 1122
   - defines how many segments can be in **transit**



Other Issues

- silly window syndrome
- slow start
- congestion avoidance
- Nagle algorithm