TCP, UDP revisited

Concurrent & Distributed Software Systems

Network Programming with sockets

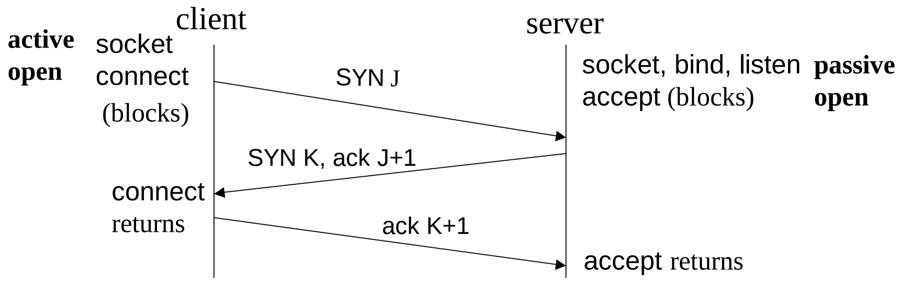
- Need to understand how TCP and UDP work in order to design "good" application-level protocols
  - critical for designing protocols that will be *scalable*
    - HTTP 1.0 does not scale well
  - when to use UDP instead of TCP
  - need to understand TCP while debugging as well as *performance* debugging

TCP

- Connection establishment
- Flow control
- Congestion control
- Connection termination

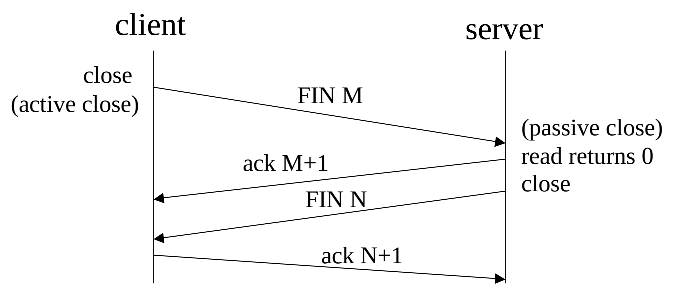
TCP Connection Establishment

- Three way handshake

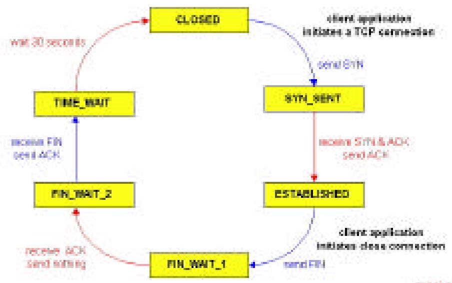


TCP Connection termination

- Four segments needed for terminating connection

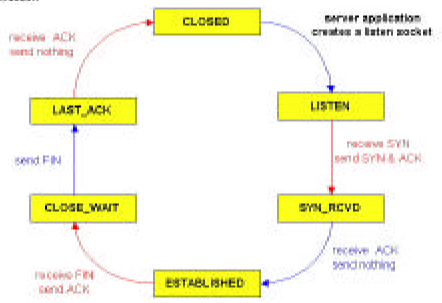


TCP Connection Management



TCP client lifecycle

TCP server lifecycle



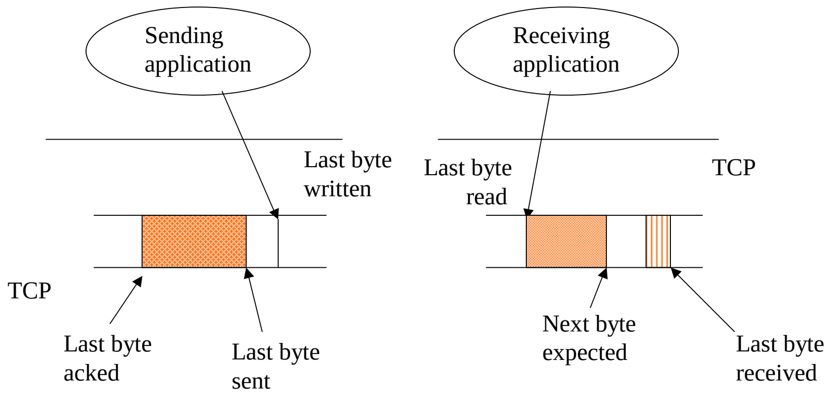
Observations

- If only purpose of connection is to send a one-segment request and get a one-segment reply there are 8 segments of overhead
  - UDP only two packets but no reliability
- TIME_WAIT state needed
  - for reliable connection termination
    - suppose last ACK lost
  - to allow duplicate segments to expire in the network
    - prevent new incarnations of connection that is in TIME_WAIT state)

TCP Flow Control & Congestion Control

- TCP uses sliding window/selective retransmit protocol for flow control
- Congestion control
  - congestion window has additive increase/multiplicative decrease
  - "slow start" algorithm

TCP Sliding Window



Receiver: Advertised Window = MaxRcvBuffer - (LastByteRcvd - LastByteRead)

Sender: Effective Window = Advertised Window - (LastByteSent - LastByteAcked)

TCP congestion control

- TCP maintains a new state variable for each connection called Congestion Window

MaxWindow = MIN(Congestion Window, Advertised Window)

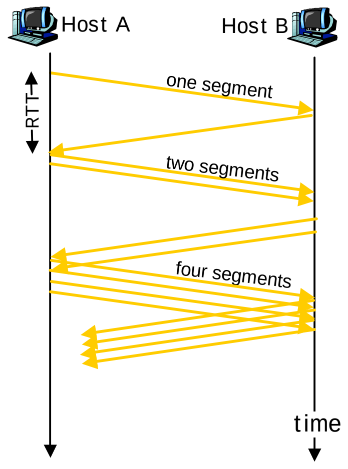
Effective Window = MaxWindow - (LastByteSent - LastByteAcked)

TCP Slowstart

Slowstart algorithm

```
initialize: Congwin = 1
for (each segment ACKed)
    Congwin++
until (loss event OR
      CongWin > threshold)
```

- exponential increase (per RTT) in window size (not so slow!)
- loss event: timeout (Tahoe TCP) and/or three duplicate ACKs (Reno TCP)



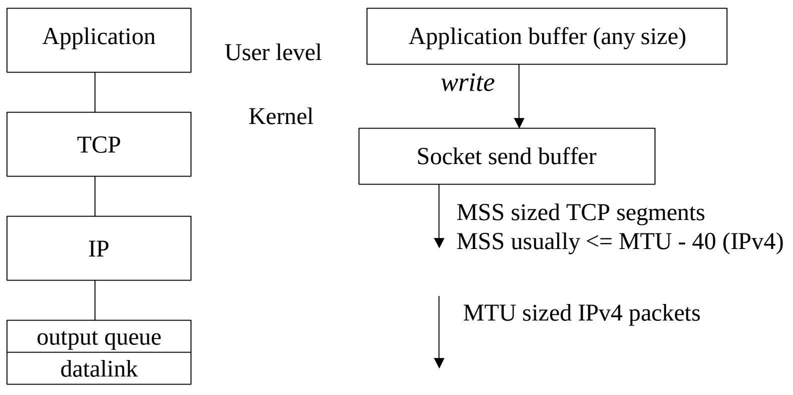
IP Datagrams and Fragmentation

- Maximum IPv4 datagram is 65535 bytes
- network MTU (maximum transmission unit) dictated by hardware
    - Ethernet 1500 bytes
- smallest MTU on path between two hosts is path MTU
- IP fragments datagram if it exceeds link MTU; reassembly done at final destination

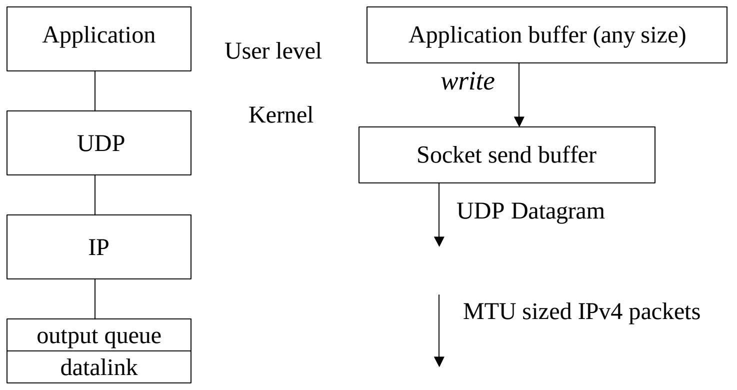
TCP MSS

- Minimum buffer reassembly size
  - IPv4: 576 bytes; IPv6: 1500 bytes
- TCP MSS (maximum segment size) announced during connection establishment
- MSS usually set to MTU - sizes of IP & TCP headers to avoid fragmentation

TCP Output



UDP Output



HTTP 1.0 revisited

- Separate connection for every document transferred
  - large overhead
  - web servers have to maintain state for every connection in TIME_WAIT state
    - can be large for busy web servers
- Slow start
  - if HTTP headers longer than MSS, client TCP needs to send two segments
  - client has to wait for first segment to be acked before it sends second segment

HTTP 1.0 revisited cont'd

- Slow start (cont'd)
  - On server side, initial congestion window = 2, so server can send 2 segments but has to wait for ack before sending any other segments
  - For files larger than two segments, slow start adds one RTT to total transaction time

UDP: User Datagram Protocol [RFC 768]

- "no frills," "bare bones" Internet transport protocol
- "best effort" service, UDP segments may be:
  - lost
  - delivered out of order to app
- *connectionless:*
  - no handshaking between UDP sender, receiver
  - each UDP segment handled independently of others

Why is there a UDP?

- no connection establishment (which can add delay)
- simple: no connection state at sender, receiver
- small segment header
- no congestion control: UDP can blast away as fast as desired

When to use UDP instead of TCP

- UDP *must* be used if the application uses multicasting or broadcasting
- UDP *can* be used for simple request-reply applications but error recovery must be built into the application
- UDP *should not* be used for bulk data transfer (e.g., file transfer)