Assignment 3- Transport Layer

11/4/2022

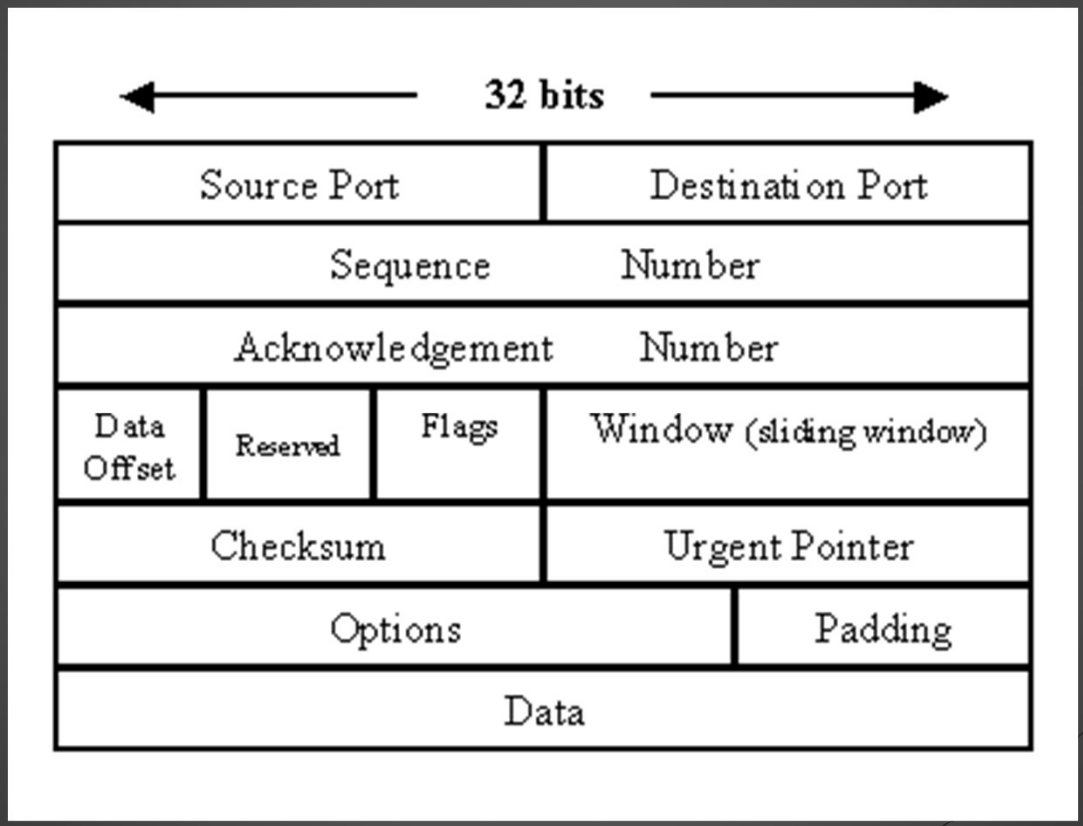
Overview

- Implement simplified TCP known as STCP
  - Runs on custom socket interface called MYSOCK
  - Assume reliable network layer where in-order delivery is guaranteed:
    - No dropped packets or retransmissions
    - No reordering
    - No timeouts
  - Does not include congestion control
  - Receives data from application and sends to networks
  - Receives data from network and sends to application

STCP Protocol

- Connection-Oriented:
  - Handshake establishing connection parameters must be performed before beginning sequential data transfer
- Data Treated as a Stream:
  - Must read all data sent by one peer, break the stream into packets and reassemble the stream on the receiving peer's side
- Full-Duplex:
  - Each connection is a pair of byte streams, one for each connection side
  - Sending on connection goes both ways
    - Active- Sending/Connection Initiator
    - Passive- Receiving

TCP Packet


32 bits
Source Port
Destination Port
Sequence Number
Acknowledgement Number
Data Offset
Reserved
Flags
Window (sliding window)
Checksum
Urgent Pointer
Options
Padding
Data

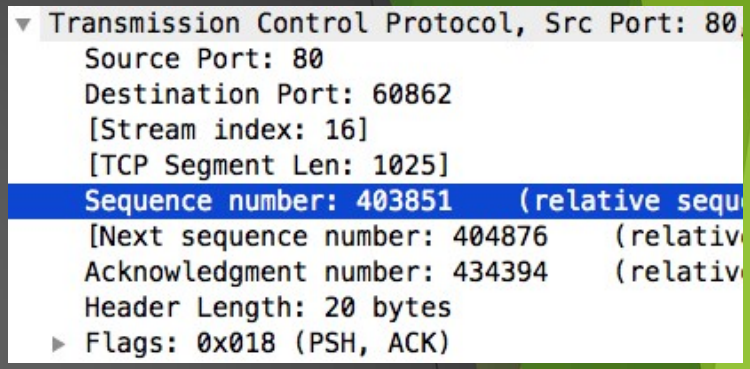
TCP Packet Structure

- Fields used by STCP Implementation:
  - th_seq
    - Sequence number of first byte in payload
  - th_ack
    - The sequence number being acknowledged if this is an ACK packet
  - th_off
    - The offset within the packet at which data begins
    - No optional data in this assignment
  - th_flags
    - Any flags OR'd together
  - th_win
    - Size of advertised receiver window in bytes

```
typedef uint32_t tcp_seq;

struct tcphdr {
        uint16_t th_sport;              /* source port */
        uint16_t th_dport;              /* destination port */
        tcp_seq th_seq;                 /* sequence number */
        tcp_seq th_ack;                 /* acknowledgment number */
#ifdef _BIT_FIELDS_LTOH
        u_int   th_x2:4,                /* (unused) */
                th_off:4;               /* data offset */
#else
        u_int   th_off:4,               /* data offset */
                th_x2:4;                /* (unused) */
#endif
        uint8_t th_flags;
#define TH_FIN  0x01
#define TH_SYN  0x02
#define TH_RST  0x04
#define TH_PUSH 0x08
#define TH_ACK  0x10
#define TH_URG  0x20
        uint16_t th_win;                 /* window */
        uint16_t th_sum;                 /* checksum */
        uint16_t th_urp;                 /* urgent pointer */
        /* options follow */
};
```

Sequence Number



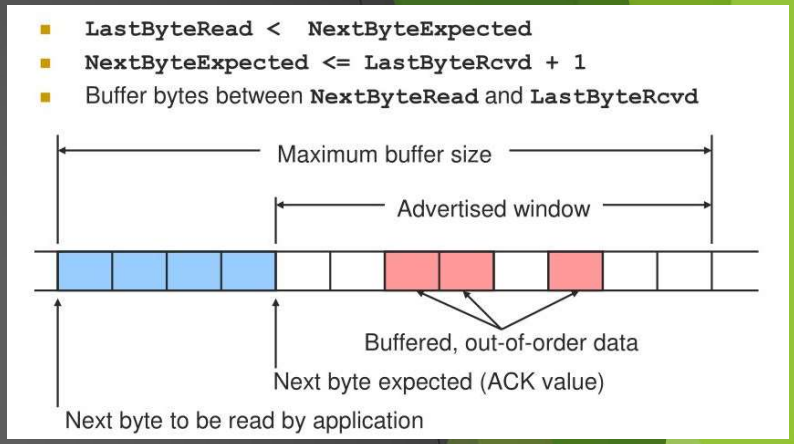
- Assigns by numbering bytes
- Rules:
  - Sequentially number
    - Sequence Number = n + Initial Sequence Number
    - n is the number of bytes in the bytestream
  - Sequence initialized to random number within 0-255 (inclusive)
  - Should be set for every packet
- SYN & FIN flags
  - Also set with the sequence numbers
  - Associated with 1 byte each of the sequence space
  - SYN- synchronizes sequence numbers of peers
  - FIN- indicates the end of the communication
- Packet Data:
  - Maximum STCP packet payload = 536 bytes
    - Stored as STCP_MSS
  - Send data as soon as available and if its within the effective window

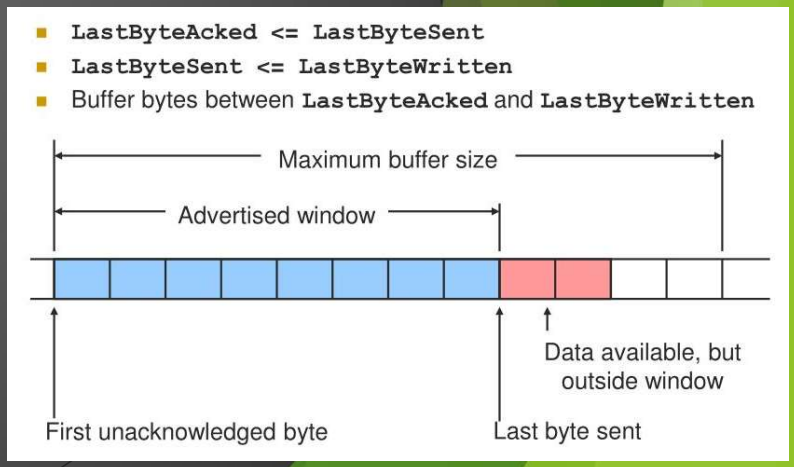
ACK

- Ensures reliable delivery
- After receiving packet, the receiver replies with an ACK packet to indicate that the packet was received
- Refer to the sequence number of the next expected byte of data
  - Example:
    - Sender sent bytes 512-1023
    - ACK reply = 1024 + Initial Sequence Number
- ACK packets contain 0 bytes of payload
- Rules:
  - Send ACK as soon as data received
    - Unlike TCP which delays sending the ACK
  - If packet has duplicate data, send a new ACK for the next expected sequence number

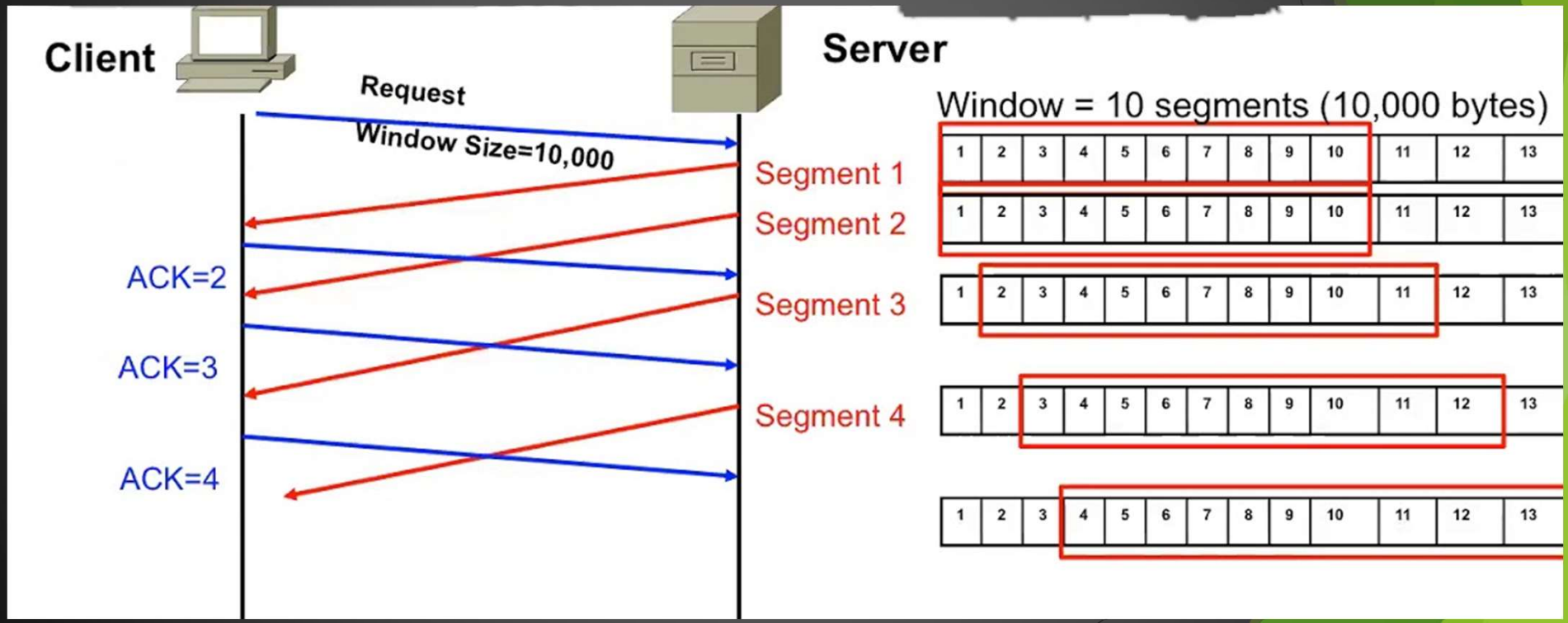
Sliding Window

- Tracks how many unacknowledged "in-flight" packets there are
  - Window size "slides" by incrementing when an ACK is received
- Receiver Window:
  - Prevents sender from overwhelming receiver
  - Sender window is equal in size to receiver window
  - Starts at last byte read
- Sender Window:
  - Starts at last byte ACK'd
- Rules:
  - Window has fixed size of 3072 bytes
  - The first byte of the window is always the last ACK'd byte





Sliding Window



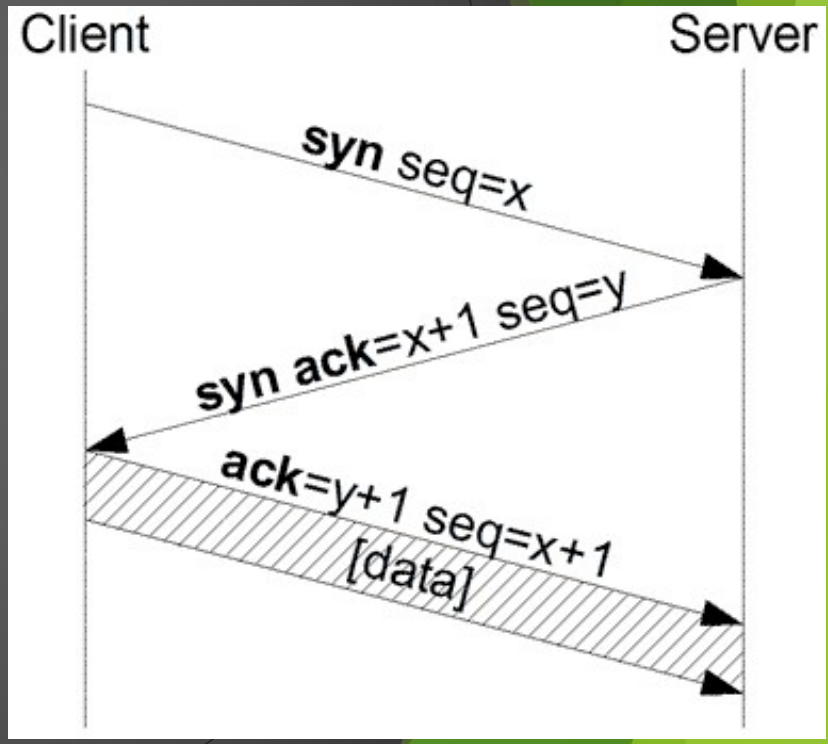
Advertised Window Size

- Receiver advertises how many bytes are left within its window
- Sender should not send packets outside of this window
- Advertised Window:
  - Size = Max Receiver Buffer – ((Next Expected Byte – 1) – Last Byte Read)
  - Can assume that the Next Expected Byte – 1 = Last Byte Received
    - Since reliable connection
- Effective Window:
  - The sending window changes with the advertised window
  - Size = min(Max Window Size, Advertised Window) – (Last Byte Sent – Last Byte ACK'd)

Connection Handshake and Closing

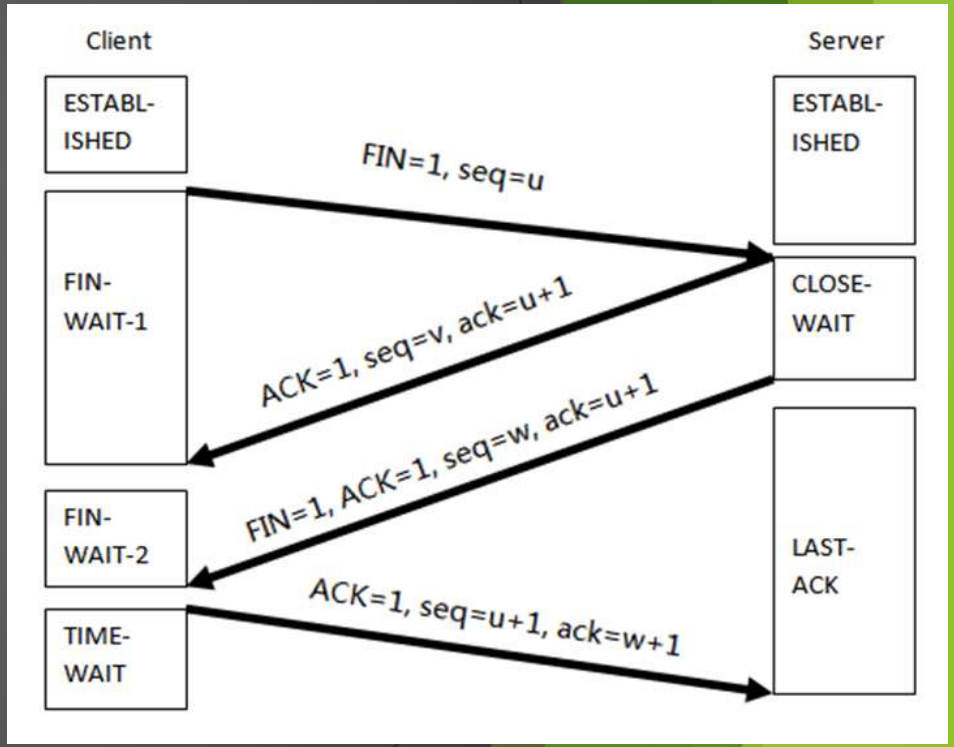
Network Initiation- 3 Way Handshake

- SYN:
  - Synchronize sequence numbers
  - Client gives server its initial sequence number
- SYN ACK:
  - Server replies with its own initial sequence number and the ACK'd client SYN
- ACK:
  - Client sends back server's ACK'd SYN



Network Termination- Teardown

- FIN:
  - Client sends packet with the flag set to indicate a teardown should begin
  - Packet may include data & receiver must be able to handle this
- ACK, FIN:
  - Server sends the ACK for the FIN
  - Sends its own FIN to indicate server teardown initiated
- Final ACK:
  - Client sends the ACK for the server FIN
  - Closes connection after sending
- Server closes connection after receiving final ACK



Getting Started

- Code to edit is in transport.c
  - See transport.h for function specifications
- Refer to stcp_api.h for functions used to communicate with network and application layers
  - Particularly stcp_network_send(), stcp_network_recv(), stcp_app_recv(), stcp_app_send()
- Test server and client are provided
  - Client sends path to a file with test input
  - Server opens file, reads input, and sends input back to client
  - Client prints what it received into file 'rcvd'
- Running:
  - Server:
    - ./server [port to listen to]
  - Client:
    - ./client –f [file path] 127.0.0.1:[server port]
  - Runs on Linux/Solaris. For other systems, use Vagrant
    - For newer MACs, use Docker as specified in the README