Computer Networks

ASSIGNMENT 3: TRANSPORT LAYER

Assignment 3 Overview

- You will be implementing STCP, a simplified version of TCP.
- STCP will run on top of a custom socket interface (MYSOCK)
- You can assume the network layer is reliable
  - No dropped packets
  - No packet reordering
  - Etc.
- No TCP congestion control

STCP Protocol Overview

- Treats data as a stream: If sender calls write() twice with 256 bytes each time and the receiver calls read() with a buffer of 512 bytes, the receiver will receive 512 bytes of available data.
- Full-duplex: Each STCP connection supports **a pair** of byte streams: one for each side of the connection.
- Specification
  - Sequence numbers and Acks
  - Sliding Windows
  - 3-way handshake with syn packets
  - Connection Teardown with fin packets

TCP Packet

- **th_seq:** Sequence number for the packet
- **th_ack:** If packet is an ack packet, the ack number.
- **th_off:** The offset at which data begins in the packet in multiples of 4 bytes. For this assignment, you can assume there is no optional data (20 byte headers)
- **th_flags:** Zero or more of the flags (TH_FIN, TH_ACK, etc.) or'd together
- **th_win:** 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 Numbers and Acks

- **Sequence Number** of a packet refers to the n-th byte in the bytestream that a packet contains as the first byte in its payload.
  - seq_num = n + <initial sequence number>
  - Initial sequence number is a random number [0,255]
- After receiving and processing a packet, the receiver replies to the sender with an ack packet, letting the client know the packet was received
- **Ack numbers** refers to the sequence number of the **next byte of data the receiver expects.**

Example:

- Sender sends 512th – 1023th in the bytestream to receiver in the TCP packet.
  - Seq_num = 512 + <initial_sequence_number>
  - Receiver would reply with ack number

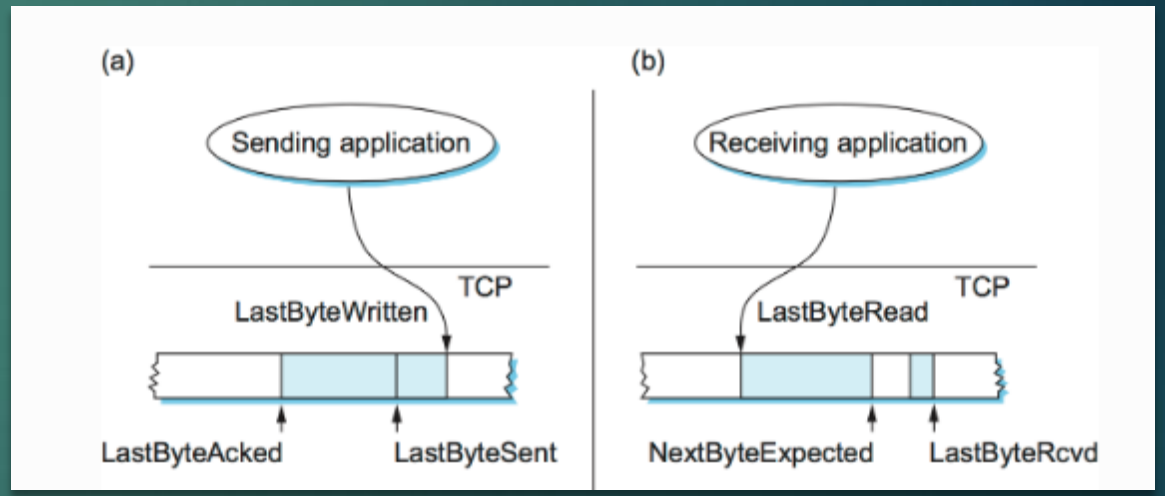
    1024 + <initial_sequence_number>

Seq Num and Acks (part 2)

- Data/Seq Num
  - Max payload size is 536 bytes (set in the global variable STCP_MSS)
  - Send data as soon as available from application if it's within the effective window.
  - ACKs have 0 bytes payload
  - SYN and FIN have 1 byte payload
- Acks
  - Unlike TCP, Acks should be sent as soon as the data is received.
  - If a packet has duplicate data, new ack should be sent for the next expected sequence number
  - For ack packets, set the seq number to the next unsent sequence number for the receiver's sending window

Sliding Window

- Keeps track of unacknowledged packets in flight.
- Sender window starts at last byte acked
- Receiving Window starts at last byte read.
- Sliding Window has a maximum size (3072 bytes)
- Rules (Textbook section 5.2)
  - LastByteAcked <= LastByteSent
  - LastByteSent <= LastByteWritten
  - LastByteRead < NextByteExpected
  - NextByteExpected <= LastByteRcvd + 1



Advertised Window Size

- **Sender should not send data outside of the receiver's receiving window!**
- The receiver advertises the number of bytes that the receiver has left in its receiving window to the client, so the client doesn't start sending packets outside of the receiver window.
- Advertised Window (textbook section 5.2):

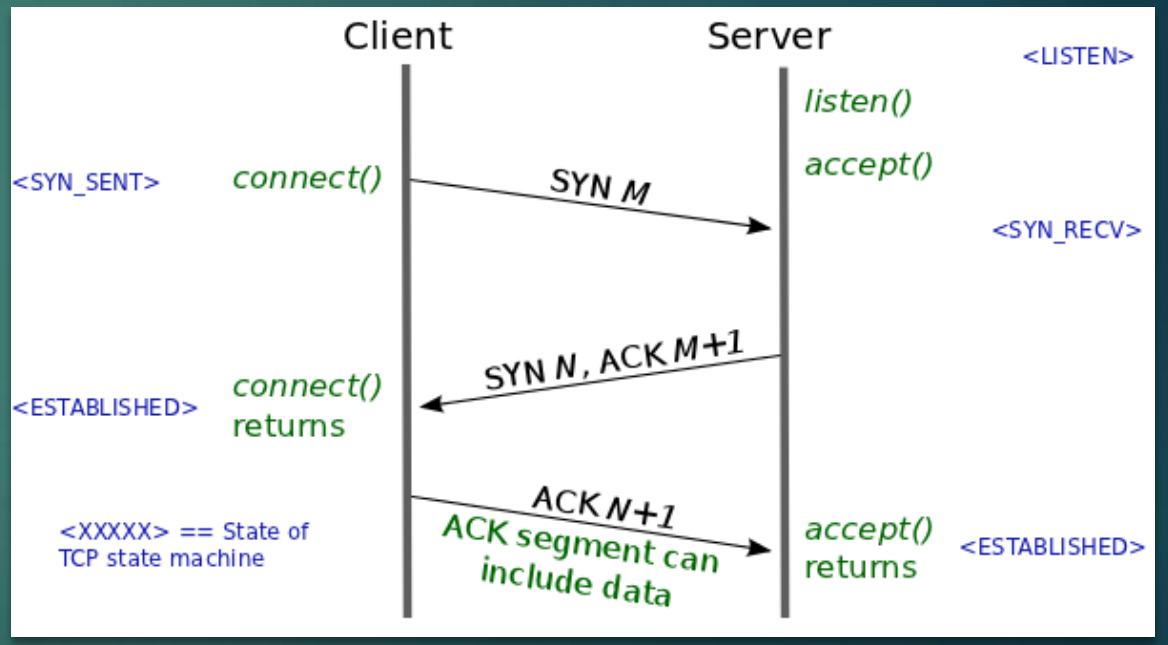
  Max_rcv_buffer - ((NextByteExpected – 1) - LastByteRead)

  - NOTE: because STCP is single threaded and runs on a reliable network, you can assume NextByteExpected – 1= LastByteRcvd
- The client's effective window is now (textbook section 5.2):

Min(Max_win, AdvertisedWindow) - (LastByteSent - LastByteAcked)

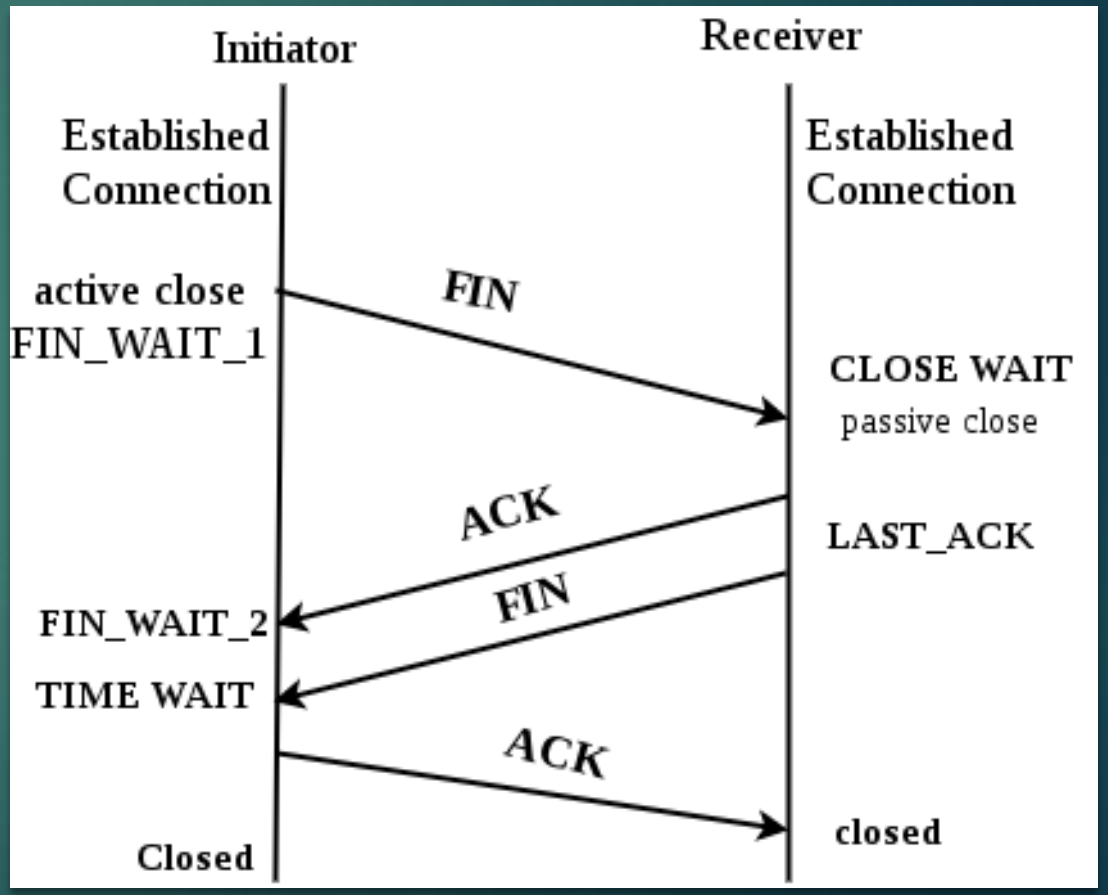
3 way handshake

- **SYN:** Client gives server its initial sequence number (M)
- **SYN ACK:** After server stores client's initial sequence number, it sends the client the server's initial sequence number in an ACK | SYN packet.
- **Ack:** Client ack's the server's SYN ACK.



TCP Teardown

- **FIN:** Client sends Server a FIN packet to start the teardown process
- **ACK, FIN:** Server ACK's the client's FIN and then sends its own FIN to the client to let the client know it's tearing down the connection on the server side.
- **ACK:** Client ack's the server's FIN packet. Closes the connection on its end.
- When the server receives the final ACK, it closes its connection



Starter Code

- **Only edit transport.c (this is the file you will submit as well)**
  - All of STCP is implemented here
  - Transport.h has the stcp packet specification.
- Stcp_api.h contains a lot of functions that STCP will need to communicate with the network and application layers.
- You are given a test server and client
  - Client sends server the path to a file with test input
  - Server finds and reads the testing file then sends the contents to the client.
  - Client outputs the data into "rcvd"

Starter Code (socket)

- My_connect() and my_accept() blocks until the handshake is finished.
- My_read() and my_write() sends/receives data from STCP.
- Look at mysock_api.c to find out more about these functions.

Final Suggestions

- Suggestions before implementing
  - Look at the functions in stcp_api.h and understand how they work.
  - Read the relevant sections of the textbook (2.5 and 5.2)
  - Design how your sliding window will work.

Good Luck!