Chiao Lu 204848946
Yuhai Li 104844760

# Project 2 Report

## Implementation Description

### Header Format

This is our header design

```
struct {
char SYN;
char FIN;
char ACK;
char flag;
uint16_t ACKnum;
uint16_t SEQ;
int reserved[4];
char DATA[1000];
} UDP_HEADER;
```

The `SYN` , `FIN`, `ACK`, `ACKnum`, and `SEQ` are self explanatory. `flag` is used to notify the client that the receiving packet contains the last piece of data. If `flag==0`, then this packet contains the last piece of the data. All other packets should set this to `1`. `reserved` is a reserved space for future modification; we can add more fields to the header without altering the size of the rest of the fields. `DATA` is a `char` array of 1000 bytes which holds the payload.

## Client Usage and Design

#### Usage

`./client <hostname> <port> <filename>`
`<hostname>` is the server's hostname; it can be either an IP address and or a domain name.
`<port>` is the port number that the server is bound to. The port number that the client is bound to is fixed to `12345`.
`<filename>` is the filename that the client wants. Client will send this filename during the three way handshake.

### Design

Client is designed using GBN method. If the arriving packet has the expected `SEQ`, then the packet is stored in the memory and a response packet is sent with `ACKnum` being the sequence number of the next expected packet; otherwise, the arriving packet is discarded and a response packet is re-sent with `ACKnum` being the sequence number of the same expecting packet.

The flow of the program is straightforward. First the client sends out the first `SYN` packet and wait for a `SYN/ACK` packet from the server. If the client then receives the `SYN/ACK` packet from the server, the client will then proceed to receiving the file. After a packet with `flag==1` is received, the client will go to shutdown stage, write the file to disk, and use the `2RTO_WAITTIME` shutdown procedure to end the program.

## Server Usage and Design

### Usage

./server <hostname> <portname>
`<hostname>` is the server's hostname; it can be either an IP address and or a domain name.
`<port>` is the port number that the server is bound to. The port number that the client is bound to is fixed to `12345`.

### Design

Server first creates a UDP socket and binds it to a port specified by the argument from command line.
The three way handshake strictly follows the rules of TCP.
The retransmission strategy is GBN-like. But instead of retransmitting all the packets within the window, when server times out, it just retransmits the first packet in the window to minimize the traffic.
In the congestion control part, just slow start and congestion avoidance are implemented in the program. To prevent the window size grows to a crazy large number, we sent a threshold for congestion avoidance to be 1.5*THRESHOLD.

## Difficulties

### Timeout Design

Designing timeout is a bit tricky because this is in some sense an asynchronous event. In the client, this problem is solved using multithreading. When a packet is sent out, a thread will be

created and timing is done in this new thread. If timeout occurs, then this new thread will retransmit the missing packet; otherwise, the thread exits.

## TC Command

Since I ssh into my remote Linux computer to do all my work, if I run `tc` command, my ssh connection becomes almost unusable. To solve this problem, I apply `tc` to `lo` instead of `eth0`.

## Debugging

Since all the packets are sent automatically, debugging is quite difficult. To counter this issue, we created a driver program that takes input from `STDIN` to set the header and data on the fly and sends packet according to the user input. This program can receive packets and display them on `STDOUT`.