CS 118 Project 2 Report
Michael Hale – 004620459
Edwin Do – 904637634

Implementation:

For our implementation of the header, we used a relatively standard approach and included the following: the sequence number, the length, and four flags for SEQ, FIN, ACK, and SENT.

```
int16_t sequence_num;
uint16_t length;
uint8_t SEQ : 1;
uint8_t FIN : 1;
uint8_t ACK : 1;
uint8_t SENT: 1;
uint8_t     : 0;
```

For the general implementation of the program, we wrote a library of functions in the file sel_repeat.c that were referenced in the two main programs server.c and client.c. Essentailly what occurred when calling the server was the server would be initialized and various metadata structures would be created in order to facilitate and implement the TCP connection. The server would then wait for a client to initiate a connection. After a connection was established, the server would then listen to the port for what the client was transmitting. Once the client finished sending, the connection would be closed and the program would terminate. Below is a list of the functions implemented in sel_repeat.c:

```
int connect_rdt(char* hostname);
int init_serv(char* hostname);
int await_connection(int meta_i);
struct pth_sp_arg* make_pack_arg(int meta_i, int sequence, int
length, int SEQ, int FIN, int ACK);
void* pth_send_packet(void* arg);
```

```
void route_packet(h_packet* packet, struct sockaddr_in*
send_addr, int sock_fd);
int fetch_packets(int udp_socket);
int read_sr(int meta_i, void *buf, unsigned int nbyte);
int write_sr(int meta_i, void *buf, unsigned int count);
void mark_done(int meta_i);
int note_thread_id(pthread_t thr);
int remove_thread_id(pthread_t thr);
void* pth_maintain_pipe(void* arg);
void finish_sr(void);
void print_packet_data(h_packet* packet);
void print_meta_data(connect_meta* meta, int meta_i);
void init_summary(void);
void print_summary(void);
void add_to_sum(m_header header);
```

Each function was used either directly in our executable program or indirectly within other functions implemented here. The interface essentially used threads that fetched packets from the UDP connection and stored it in a metadata structure where it handled and noted incoming data and acknowledgements.

We were given a timeout value of 500ms. Our implementation was to wait the given amount of time and then check if the packet had been successfully received, indicated by an ACK or the sequence number being properly increased, if it was not successfully received by then the packet would be retransmitted.

Difficulties:

A significant portion of the difficulty associated with this project was in the specific details of getting a working TCP-like connection. A large amount of the early development and bug-fixing for this project was spent getting some of the basic functionality of the connection working, such as sending a single packet message. To properly emulate a TCP connection required an extensive amount of additional features. We had developed a large library of

functions that were used to essentially communicated between the application layer and the transport layer.

Once we established the basics of the connection, the next major difficult we faced was in the open-ended design of the implementation. In contrary to the previous project, a lot of the implementation was a matter of choice versus a specific guideline to follow, this allowed for more creativity in coming up with a working product and solution, however a lot more time was spent researching various documentation and examples to properly figure out the implementation we used.

Another difficulty we faced was that there was a significant amount of interconnection between the various portions of the server and client as well as a large number of edge cases that all needed to be considered, because of these, there was an extremely large amount of bug fixing that we needed to perform after implementing any feature of this project as well as when making any adjustments to existing features, it was hard to fully consider which other portions of the code would be affected, which led to more extensive bug fixing.