CS118 Lab 2 Report

Aditya Raju UID: 104407651
Srikanth Ashtalakshmi V Rajendran UID: 804478782

Extra Credit Implemented: We implemented Fast Retransmit and Recovery, and client side buffering (All the extra credit is implemented in the same file as the base project)

Note: Data transmitted is saved in a file called result.data

Specific instructions to build and run our code:

Run make, which should make the executables “server” and “client”. Use these executables to test the base project AND the extra credit.

High level description of our server and client:

Client: the client initiates the communication by sending an SYN packet. It then waits for a while, and if it doesn’t hear back, it retransmits the packet. The, after the SYN-ACK is received from the server, the client goes into “pure ack mode”. Basically it keeps track of the expected sequence number from the server, and buffers packets that have a greater sequence number than that. Then, when a sequence number fills a hole, it cumulatively ACKS. Therefore, our client **implements a client side buffer.** This goes on until a FIN is received. Then the client sends a FIN ACK and a FIN message to the server, and waits for a FIN-ACK message. If it receives a FIN ACK message it closes, if not, it resends the FIN ACK and FIN messages (this can happen a total of 4 times, after which the client assumes the connection was closed and terminates).

Server: The server takes in a file and divides the file into data packets. It then receives a SYN from the client and sets the headers for all the data packets based on the sequence number that the client sent. After, it sends a SYN-ACK and will retransmit until that is ACKED. From there we start sending the data packets. We start in slow start and once we hit the ssthresh, we go into congestion avoidance. If there is timeout, we go back to slow start and a window size of 1 MSS. We implemented fast retransmit, so if there are three duplicate ACKs then we set cwnd to half its value and set ssthresh to half the cwnd as well, and go into Congestion Avoidance. Lastly, we send a FIN and wait for the FIN-ACK. After that we see a FIN from the client, and send a FIN-ACK, and wait for around 100 seconds to close the connection.

Problems we ran into and how we solved them:

1. Client side buffering -> It was hard to deal with wrap around sequence numbers. Solved it by drawing situations out on the whiteboard, and representing where to place packets based on sequence number mathematically.
2. Understanding the connection tear down process. The slides showed two different things, so we used outside resources to see how to do it.
3. How to manipulate the congestion control window -> read the textbook for more clarification
4. Having trouble with connection between virtual machines -> have to periodically ping each other to “reset” the connection
5. Speeding up our client and server using correct timeout implementation

How we tested our code:

We tested our code incrementally. We first just tried to send one packet, and receive an ACK for it. We then moved on to sending data packets for file and acking them over a reliable channel. We then added connection set up and tear down procedures. We then added packet loss and packet reordering and saw how it performed. Lastly, we tested our code against the large.txt and small.txt files, with 20% packet loss. This worked perfectly. Throughout all of these tests, we would look at the output to console to see if ACKs and congestion window resizing was being done properly or not.

Contributions of each team member:

Srikanth: Server side operations, and fast retransmit and recovery extra credit
Aditya: Client side and client side buffering extra credit
Both: Helper/Utility functions and classes used by the server and client