Tony Luo

A98063711

CSE 123

The first step was to create the header and the footer. Since the communication is unidirectional, and no sender will become a receiver, we can use the same unsigned char for the sequence number and the acknowledgement number. In addition we use 2 unsigned ints for the sender id and the receiver id and another unsigned char for the CRC footer given us a total of 6 bytes for the header and footer.

Next step was to edit the sender and receiver to implement the sliding window. For the sender we added an unsigned char for the sequence number and another unsigned char for Last Acknowledgement Received. In addition we created an array to hold the messages that are waiting for Acknowledgement and a counter to see if the array is full. On the other hand, for the receiver we implemented a buffer to receive the packages that are out of order and an unsigned character to keep track of the Next Frame Expected.

The way I implemented the sliding window protocol is that the sender will send as many messages as it can fit on its buffer and will wait for the receiver's acknowledgment to clear the buffer. The receiver will always sent an acknowledgement for the Next Frame Expected and will increase the Next Frame Expected every time it receives the frame that it is expecting. In case the receiver gets a frame that is not within its window, the receiver will just sent an acknowledgement for the Next Frame Expected.

The sender on the other hand, will only remove the message from the buffer if it receives an Acknowledgement that is greater than the Last Acknowledgement Received. Upon the receipt of the Acknowledgement, it will assume that the receiver has already accepted all the frames leading to the one that just got Acknowledge. While doing that, we clear all the frames in the buffer that were in between the Last Frame Acknowledged and the Next Frame Expected. In the case that the sender doesn't received an Acknowledgment from the receiver, it will keep resending the buffered frames based on each of their time outs.

Furthermore, we used a CRC8 to determine whether a frame got corrupted during its traversal. So if either receiver or sender realizes that a frame got corrupted it will automatically drop it and wait for a retransmission.

Finally, in the case that a message is bigger than what our frame can support, we split the message into partitions that can fit the payload. The sender will place the correct sequence number and using those, the receiver can display the complete message in the correct order.