**Documentation**

**Stop and Wait Implementation:**

**Client:**

The client will begin by sending the message first **(using a for loop that will continue until maximum number)**, then wait for the server to reply with an acknowledge message. Once the message is received, it will match it with client’s sequence number **(the ‘i’ in our for loop is our the sequence number)**. If the sequence number and the return acknowledge number are equal, then client will increment its sequence number and transmit another packet. If they happen to be not the same, then a packet drop must occur. In the event of a packet drop, client will not increment its sequence number, instead it will start a timer and wait for 1500 usec. Once the timer reaches its limit, we will count the missing packet **(decrement ‘i’ by 1 to reset the sequence number, this is necessary because a for-loop is linear)**, and resend the same packet to the server.

**Server:**

When the server starts, it will listen to the socket and wait for data to come in. It too will use a for loop to send back the sequence number. We want both client and server to use the same for loop because the client will send a sequence number and the server need to reply with corresponding number. Within the for-loop, there is another while loop acting as a trap state if there are no data coming in. Once data comes in, it will evaluate if it is the correct corresponding sequence number. If sequence number do not match, then the server will stay in the trap state. We want the server in trap state to ensure that the client will receive the correct sequence number. If it is correct, then it will transmit an acknowledge packet back to client and break out of the while-loop, for the for-loop to move on until the maximum value (20,000).

**Sliding Window Implementation:**

**Client:**

To begin the implementation of sliding window, first we declare 2 important variables. Int acked – the number of acknowledged sequence numbers. Int nonAcked – the number of not acknowledge sequence numbers. Similar to stop and wait, we will use a for loop and the int i will be our current sequence number we want to send to the server. Since sliding window can keep sending sequence numbers up to the window size, we have to mark each sequence number with a timer to keep track in case of a time out. We will continue to send to the server until all the number of non-acknowledged sequence number reaches the size the of window. Once that happens, we will be in “quick recovery mode”. We will listen to the socket and hope for the acknowledge packet to come through. If it does, then we can decrement the nonAcked variable by each corresponding acknowledge number and increment the variable acked to keep track of all acknowledged sequences number. In the event of incorrect acknowledge packet, once the timer reaches 1500 usec, we will have to resend all the packet that was unacknowledged. First, we need to tally up all packets we haven’t sent into count. Then, we will set ‘i’ to value of acked because we don’t want to resend already acknowledged packets. This will continue until all sequence number from 0 to 20,000 are sent and received with corresponding acknowledge number.

**Server:**

Following similar design as the rest of the program, we will use a for-loop and its variable ‘i’ as our sequence number. To ensure that we send back the correct acknowledge number, we will set a while loop inside the for-loop. This will trap the server and the only way for it to get out is if it receives the correct sequence number. The server will continue to listen on the socket and wait for packets to come through. Once a packet arrives, it will respond with ack, then it will check the if the sequence number from client matches the server’s sequence number. If it doesn’t, it will go back and wait for another packet, until a match finally comes. Once the sequence numbers matches, we will break the trap state and increment the sequence number of the server. This will continue until the maximum value (20,000)