Calvin Liu 804182525
Matteo Vesprini-Heidrich 304205341

The way we implemented it was to create a packet structure with a type, sequence number, size length, and the length of the data. By doing this, we could send each structure across the channel with the UDP socket programming functions provided in the C library.

For the server, what we did was read in the file and put it inside a packet structure. When the packet structure was 1024 bytes long, we would send the packet. We would keep doing this based on the window size. Once we sent all the packets, if we detected that the client sent an ACK packet based on the packet structure type, then we would move the window up. We then implemented the chance of corruption and packet loss. To do this we used the random function to produce probabilities between 0 and 1. If there was a problem then we would not do anything. Next we implemented a timer function to time out when some kind of packet was not received on time. If a packet was received on time then it would do what it was supposed to do but if the packet or ACK did not arrive somewhere in time then we should send the whole window size again and just slide the window based on the highest ACK number.

The client begins by binding to a socket and reading the command line arguments into variables. Then, it sends a request to the server containing the filename of the file we wish to download. Then begins the main loop of the program that reads the data packets the server is sending and writes them to a file. First we check for a data packet with a certain probability of loss or corruption dictated by our command line arguments. If the packet we receive is corrupted then we resend an ack for the last in-order packet we received in compliance with go back n. If the data packet makes it through the dangers of the network unscathed, we check if it is in order. If it is, we write it to our output and ACK the sequence number that we expect next from the server. Otherwise we discard the packet and send an ACK with sequence number based on the last in-order packet. Once we receive the last ACK packet we exit our data loop and begin termination of the connection. We send a FIN, await a FINACK, await a FIN, and finally send a FINACK and close the socket to terminate the connection.

We faced a lot of bugs and to solve them we just used GDB and drew out the diagrams on how the implementation should be. http://www.ccs-labs.org/teaching/rn/animations/gbn_sr/ was a big help in showing us how Go-Back-N really works. Our design implementation tried to mimic that. A big problem that we faced was not knowing how to replicate a real Go-Back-N situation because we had to implement the probability of the packet being dropped and the packet being corrupted. Since we had to implement that on our own, we also had to implement the print function and everything else after simulating this drop and corruption, which really cannot happen since we are doing all of this on the same machine.

To create the necessary object files, you only need to type make after getting the Makefile from unzipping the file.