I completed this project for an assignment in school. You can find the assignment specifications in the other PDF file in this folder, but I'll give a brief summary here.

My job was to implement two separate programs (Homeland and Embassy), where Homeland's goal is to send the contents of a file to Embassy. The challenge, however, is that Homeland must send its data by way of a middle program called Relay, which will simulate issues that might occur in the network such as corruption of data or loss of packets. The Relay program was provided to me by the professor. Despite the interferences from Relay, Homeland and Embassy must ensure together that the entire file is transferred correctly from source to destination. Thus, I had to implement a reliable data transfer protocol at the application layer over UDP.

In addition to the code, I had to write a README explaining the protocol that I used. All the following text in this file is the README that I submitted with the code when I turned in the assignment.

/* Caleb Shere 2-49327691-9 */

The protocol that I implemented is essentially the Selective-Repeat protocol. I will explain the general design of the protocol, followed by the operation of Homeland and Embassy separately.

General Design:

Each packet consists of the data from the file, the packet's sequence number, the checksum, and a boolean signifiying whether this packet is the final packet of the file. Homeland stores for itsef a queue containing the packets that have been sent but not yet acknowledged. Embassy stores for itself a queue containing packets that have been received but are waiting for an earlier packet before being written. Each queue follows a strict window of sequence numbers, so that the range of sequence numbers in the queue is always from base to base + QUEUE_SIZE. Thus, additional packets can only be added to the queue when the base packet is removed from it (and thus the base number is updated).

In this way, Homeland pipelines packets to Embassy and waits for them to be acknowledged. When an ack arrives, Homeland notes a specific packet acknowledged and then may be able to add more packets to the pipeline. Any packet that is not acknowledged 3 seconds after being sent is sent again. On Embassy's end, Embassy simply sends an ack for every correct packet received, stores them in a queue, and then writes them to the output file in the correct order based on their sequence numbers.

Homeland:

Homeland stores its own packet queue of size 10, which is made up of packets that have been sent but not yet acknowledged. It also stores the important values SOCK_TIME (the time that a socket receive function will wait before "timing out") and TO_READ (the size in bytes of each block read from the input file).

Homeland's operation is as follows: as long as the last block of the file has not been read yet, the program gets some number of packets (this number is initialized to QUEUE_SIZE and I will explain in a bit how it is updated throughout execution). It then sends these packets to RELAY, and the send_packets function returns how many additional packets should now be retrieved from the file. After the last block has been read from the file and sent, the program waits until all packets in the packet queue have been acknowledged. At that point, it terminates.

The send_packets function takes each packet given to it; sends the packet to RELAY; and then records a tuple in the packet queue consisting of the packet, the packet's sequence number, and the time at which it was sent. It then iterates over the packet queue and resends any packet that was sent more than SOCK_TIME seconds ago (this packet has timed out, and thus may have been lost, so it needs to be resent). It then calls the receive_acks function.

The receive_acks() function is very important. First, it tries to receive data from RELAY. If the socket receive function times out, it resends all packets in the packet queue and then calls itself again. Else, if data is received without timing out, the function reads through every ack received and - if the ack is not corrupted - it removes the packet that is being acknowledged from the packet queue. If the packet acknowledged was the "base packet" (meaning it had the lowest sequence number in the current window and occupied the first index of the packet queue), then the base is moved to the unacknowledged packet with the lowest sequence number. It then returns the number of values that the base moved - this is the number of additional packets that the queue can now take

from the file.

Embassy:

Embassy also stores a packet queue of size 10. This queue is made up of packets that have been received but can't be written yet because they're waiting for preceding packets in the file to arrive so that the file will be written in order. The operation is as follows.

As long as base (the lowest sequence number of the packet we're still waiting for) is less than or equal to last_num (the sequence number of the last packet in the file), Embassy receives a packet, and if it is not corrupt and not already in the packet queue, it records the packet in the packet queue. If the packet's sequence number is equal to base, Embassy writes the packet to the output file, as well as any previously received packets with consecutive sequence numbers. The base is then moved by the number of packets written.

When the last packet is received, Embassy records its sequence number, and when base becomes greater than that sequence number, it means we have written the last packet and all packets before it, so we're done. Embassy then sends a series of "done" acks to Homeland just in case its last ack or two was lost and Homeland still doesn't know that we finished. It then terminates.