Students:
I-Chun Liu - 504434209
Jason Tieu - 304047667

Implementation description

- Header format
    - There are 3 things in our packet header: type, sequence number, and size.
    - Type indicates the type of this packet.
        - 0 - Request
        - 1 - Sequence Num
        - 2 - ACK
        - 3 - FIN
        - FIN is used to signal both the client and server that all the packets have been successfully received or sent out.
        - ACK is used to indicate which packet has been acknowledged by the client.
    - Sequence number is used to determine the ordering of each packet.
    - Size is the size of the data. Since data size in each packet can be different, we added this variable to our packet header. Normally, the data size of each packet from the server is 1000. However, the data size of the last packet from the server is not necessarily 1000 unless the total number of bytes of the entire data is divisible by 1000. On the client's side, all packet's data size is 0 with the exception of the first packet because it contains client's requested filename.
- Messages
    - For both client and server, the console outputs the timestamp that the event happens, along with the sequence number, type, and timeout/retransmission occurrences. In addition, the console outputs a random probability, which is calculated using rand() and RAND_MAX, and the user's assigned probability. Upon packet loss or packet corruption, the console outputs the message "simulated loss" or "simulated corruption" with the packet's sequence number.
    - For both the server and client, output messages are in the format:
        - SENT: Type X Seq Y Size Z **TimeStamp:#ms
        - RECV: Type X Seq Y Size Z **TimeStamp:#ms
            - X is the packet type:
                - 0 - Request
                - 1 - Sequence Num
                - 2 - ACK
                - 3 - FIN
            - # is the number of millisecond that has passed since the start of this program
        - elap TIMEVALUE
            - TIMEVALUE is the elapsed time
        - Simulated loss SEQUENCE

                - SEQUENCE is the packet's sequence number.
            - Simulated corruption SEQUENCE
                - SEQUENCE is the packet's sequence number.
        - For the client, output message is in the format:
            - IGNORE SQ: expected SQ2
                - SQ is the the packet's sequence number, which is being ignored by the client.
                - SQ2 is the expected packet's sequence number.
        - For the server, output message is in the format:
            - Sliding window to NUM1/NUM2
                - NUM1 is the current base number in GBN.
                - NUM2 is the total number of packets.
- Timeouts
    - We choose the timeout to be 1 second because waiting longer than 1 second appears to be too long of a wait. However, waiting less than 1 second is probably not enough because the packet may have been delayed and not lost. We don't want to resend a packet if it's delayed rather than lost or corrupted.
- Window-based protocol
    - We decided to implement Go-Back-N protocol, it is derived from the ARQ (automatic repeat request) protocol. The sending process keeps sending a number of frames specified by the cwnd even if it doesn't receive an ACK from the receiver. The receiver process continually keeps track of the sequence numbers of the next frame it expects to receive, and sends that number alongside the ACK. Frames that do not have the exact sequence number expected are tossed away by the receiver. An ACK for the previous correct in-order frame is resent. Once the sender has all the frames in its window, it will find the frames since the first lost frame and go back to the sequence number of the last ACK it received and fill the window starting with that lost frame and repeat the process.
- Emulate Packet Loss and Corruption
    - First, we take the probability of packet loss and packet corruption from the user. For every received packet from the client, we will calculate a random probability using rand() / RAND_MAX. If the random probability is less than or equal to the user's probability, packet loss or packet corruption will occur.

Difficulties that we faced and how we solved them

One of the difficulties that we faced was figuring out why UDP was not working between the client and the server. After 30 minutes of debugging, we realized that instead of having "SOCK_STREAM" in our socket function call, we should have "SOCK_DGRAM". Specifically, "SOCK_STREAM" is only used in TCP, and "SOCK_DGRAM" is only used in UDP.