My UDP and TCP implementations are quite similar in overall implementation. They are both written in python using python's socket programming My UDP server can establish a socket and bind it to the first available host, or the local host independent of the client. The UDP client can establish a socket and wait for input from the user to send to the server after requesting a server address name and loss rate as input independent from the server. When both are running, regardless of which program runs first, the client begins waiting for input from the keyboard (after specifying the server name) to send to the server. After receiving this input in the form of a string, it is encoded and sent to the server over the UDP socket. On the server side, the request is read from the buffer and is decoded. The resulting string is parsed and the input is checked for validity. If the input is of the valid form: x1 oc x2, it is converted to integers and the appropriate operation code to be evaluated. After evaluation the resulting integer is encoded and sent back to the client with a 200 OK status code. If the input is invalid the server sends an invalid response back with an accompanying 300 status code. After the server sends the response, it begins waiting for a request from the client once again. The client receives the server's response and parses the decoded message to check the status code. If the status code is 200 OK, the result is displayed on the client, otherwise the status code is 300 and there was an error with the input and a warning is displayed on the client. After dealing with the response, the client begins waiting for input to send to the server again. This input is sent to the server with 100% probability but is received by the server with probability determined by the input at the beginning of the client program. If the request is received by the server within an initial .1s it is handled. Otherwise an exception is handled on the client side while waiting for a response from the server and the timeout is doubled while the request is resent. This process continues until the timeout exceeds 2s, in which case the timeout exception is thrown and the client program exits. The only other way to exit the program is to request "quit" which closes the sockets and exits both programs.

The TCP client and server are implemented quite similarly to the UDP implementation. The main difference is that it is reliable and thus does not suffer from packet loss. Every request is seen by the server and every response is sent back to the client. The client cannot run independently of the server because the connection is refused on the server side as it is not established. The server can run independently of the client and a socket is established. The same process is then followed for sending requests, handling them and sending a response back as detailed in the UDP description above.

During my implementation of the timeout in the UDP client, I encountered some issues. I first tried to install a threading_timeoutable decorator for python using pip, but the execution didn't seem to work. Threading and multiprocessing with a timer using python's time module also wasn't working properly. My windows machine didn't support UNIX signals so I was limited to what I could do. I eventually found a settimeout function I could call on socket objects. This coupled with python's try except block handled the timeout exception quite nicely.

To actually run the program python needs to be installed on the client and server hosts. Export the client-server.zip file to a directory of your choice. Navigate to the directory using a terminal and do so again on another terminal. Run the XXX_Server.py and accompanying XXX_Client.py in either terminal. Remember to follow the requirements for running the programs independently of each other and the input the initially require as detailed above.

If either of the programs is exited manually by the user, you should restart them based on the instructions listed above.