Dylan Schneider, Roark Myers, Josias Polonia

November 3, 2019

Network Design

Vokkarane

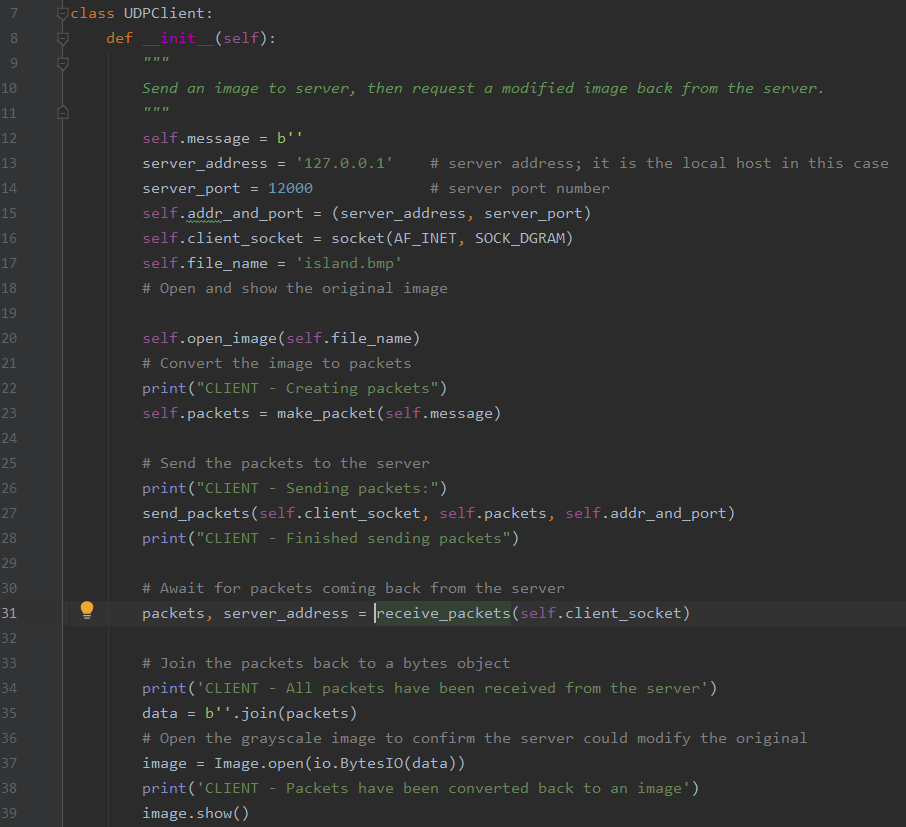
Design Document: Phase 3

**Purpose of the Phase:**

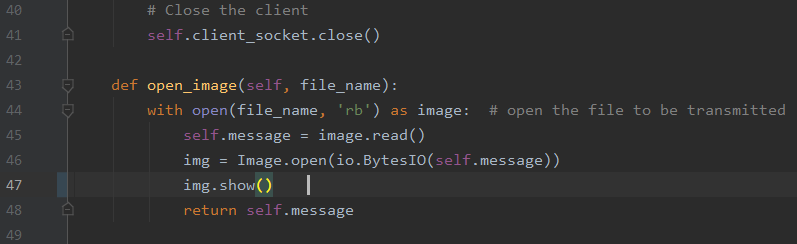
The purpose of the third phase is to implement the RDT 2.2 method (reliable data transfer) which now allows bit errors and error detection; now positive acknowledgements are used to notify the sender whether or not the sent packet had any errors and it received it. Not only must the receiver clearly state the sequence number of the ACKed packet but a checksum must indicate that the packet isn’t corrupt. Two ACKs are received for the same packet if the receiver didn’t get the receive the packet correctly.

**Code Explanation:**

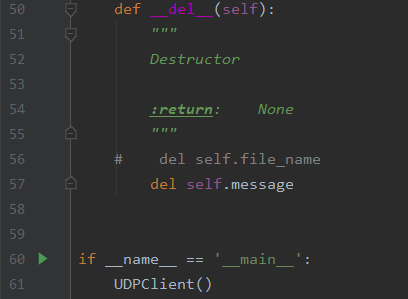
*UDP Client:*



After the initial function declarations, the overall task of the Client class is to send the image to the server and ask for it to be sent back after received. The server’s address and port number is stated, then the client socket is initialized; this covers lines 12-17. Once the image is opened, it is converted to packets in line 23 (make\_packet function); the packets are then sent to the server in line 27 until completion. After the server performs receives the packets, they are received back and stored/converted back to bytes (lines 31-35). Line 37 is just used to verify that the server sent back the original file by displaying it.

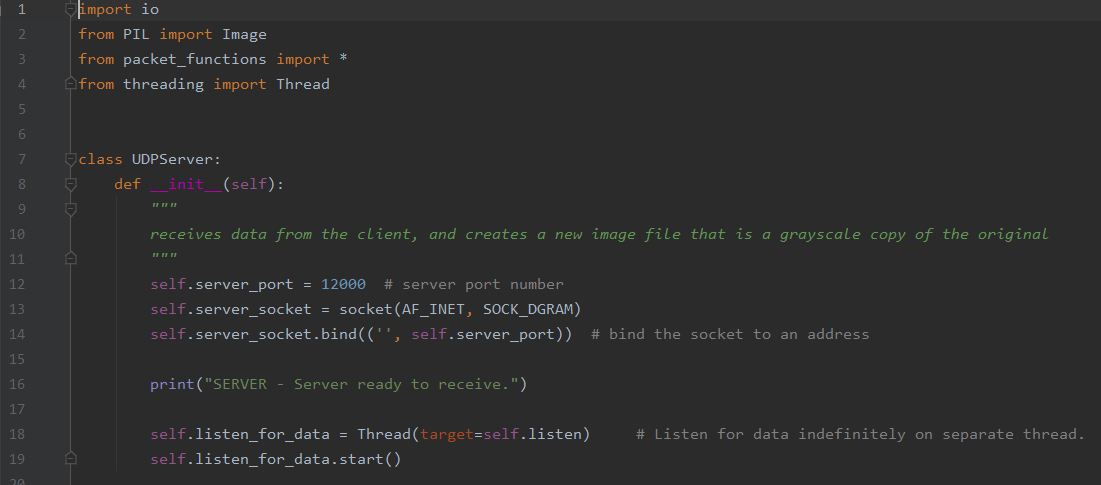


Now we look at the open image function which was called within the last definition. Line 44 simply opens the file to be sent by reading it into self.message, and opening the image into bytes.

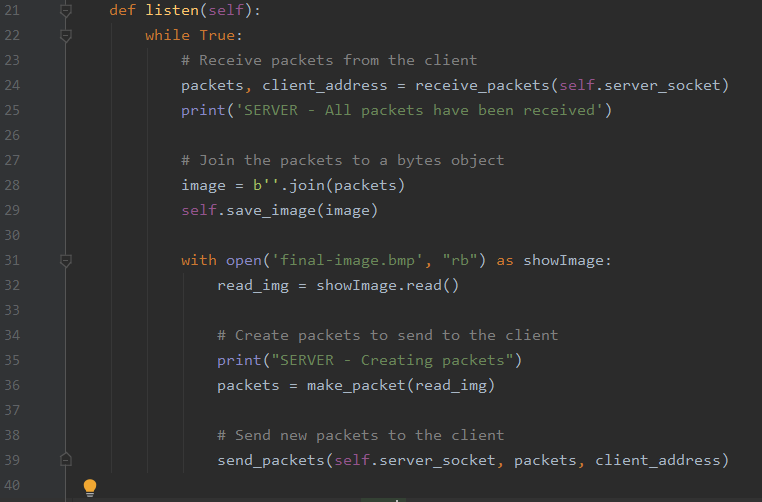


This section declares the destructor which then gets called once all references have been deleted.

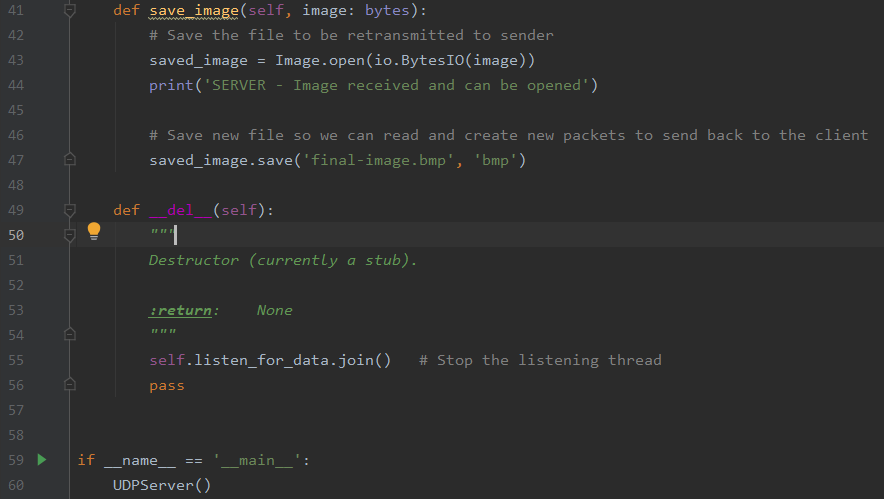
*UDP Server:*



The Server states its port number and its socket; line 14 attaches the socket to the port number. In the while loop the server listens for incoming data.

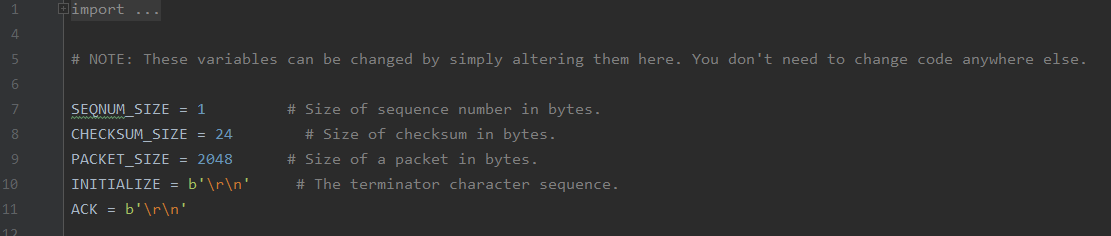


The server receives packets sent from the client and stores them until all have been received. They are combined into bytes to create a full image. Line 36 begins to make packets, this time to send back to the client. Self.save\_image is used to save the image on the receiver so it can be sent back to the sender. Line 39 uses the socket, created packets and receiver address to send the packets over.

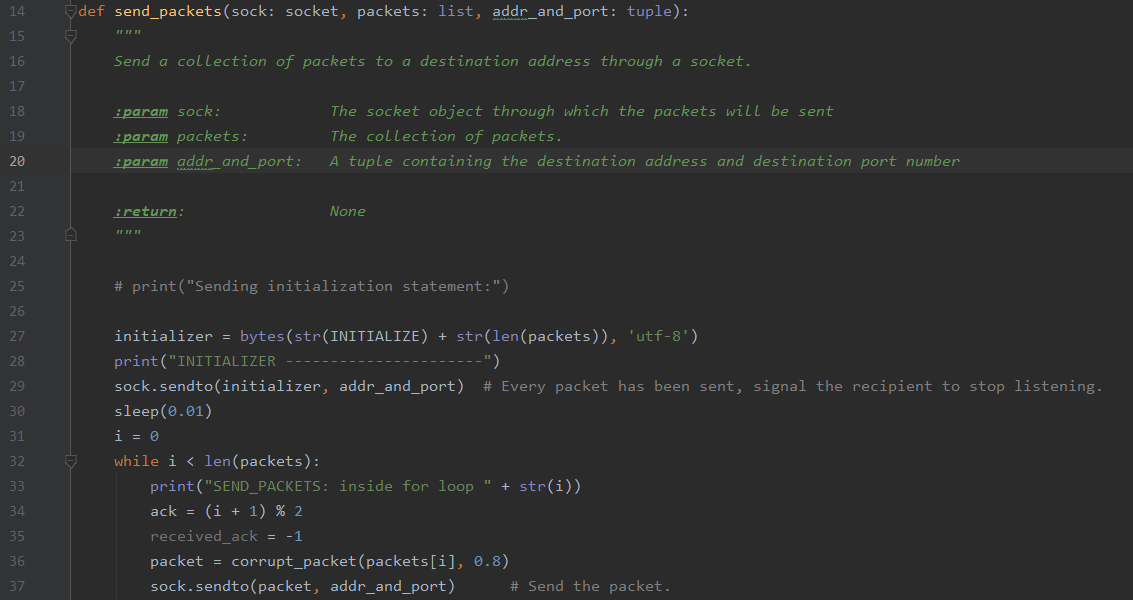


The save\_image function opens up the received image and stores it into bytes; line 47 then saves the image to be read and prepare to create new packets and send them back to the receiver once more.

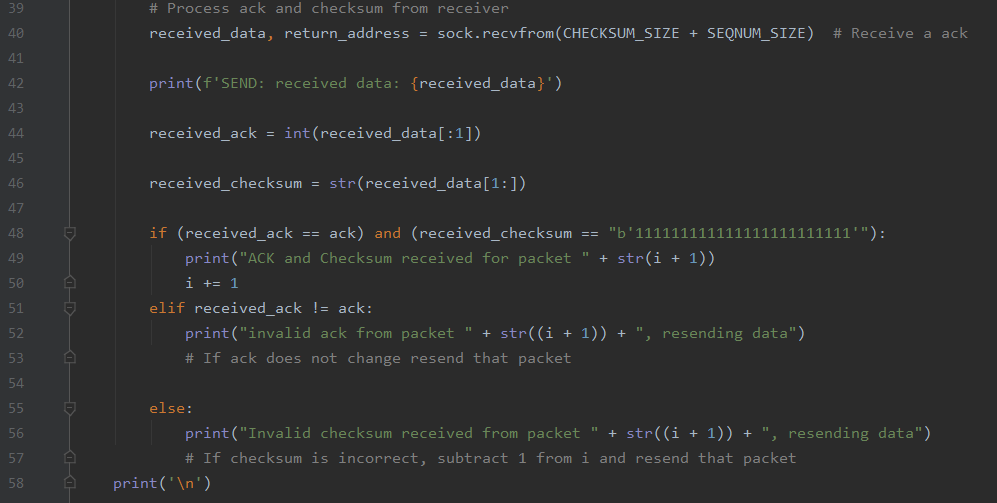
*Packet\_Functions*



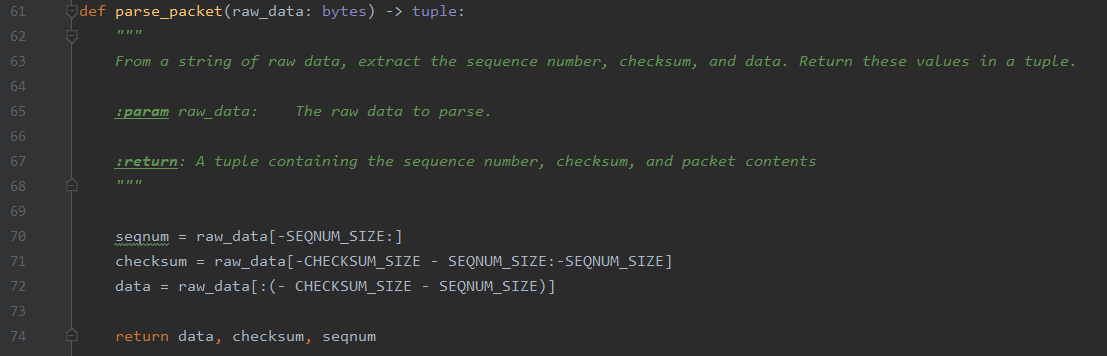
This section states the sizes and initializer character sequence.



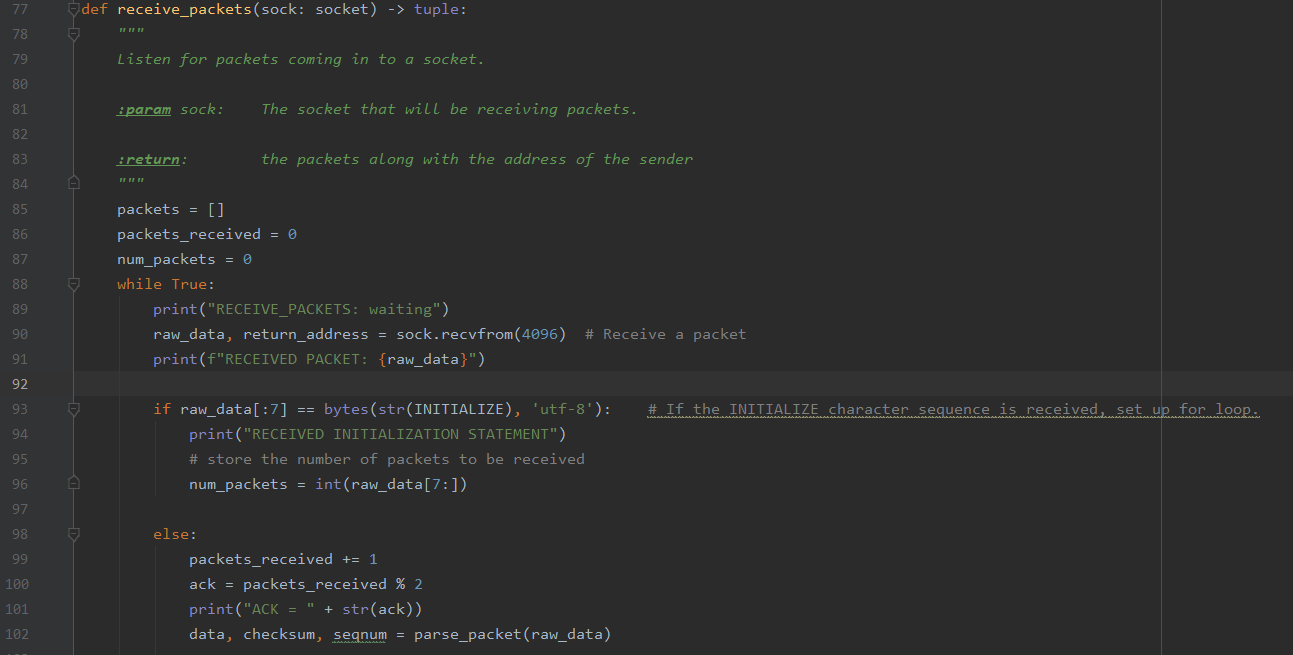
The send\_packets function, through a socket, sends a bunch of packets to desired address. Line 27 creates an initialization and then sends all the packets; This tells the receiver how many packets to listen for. After the ack declaration, received\_ack, packet is set equal to -1 by default and then gets overwritten by the ack we get from the receiver.



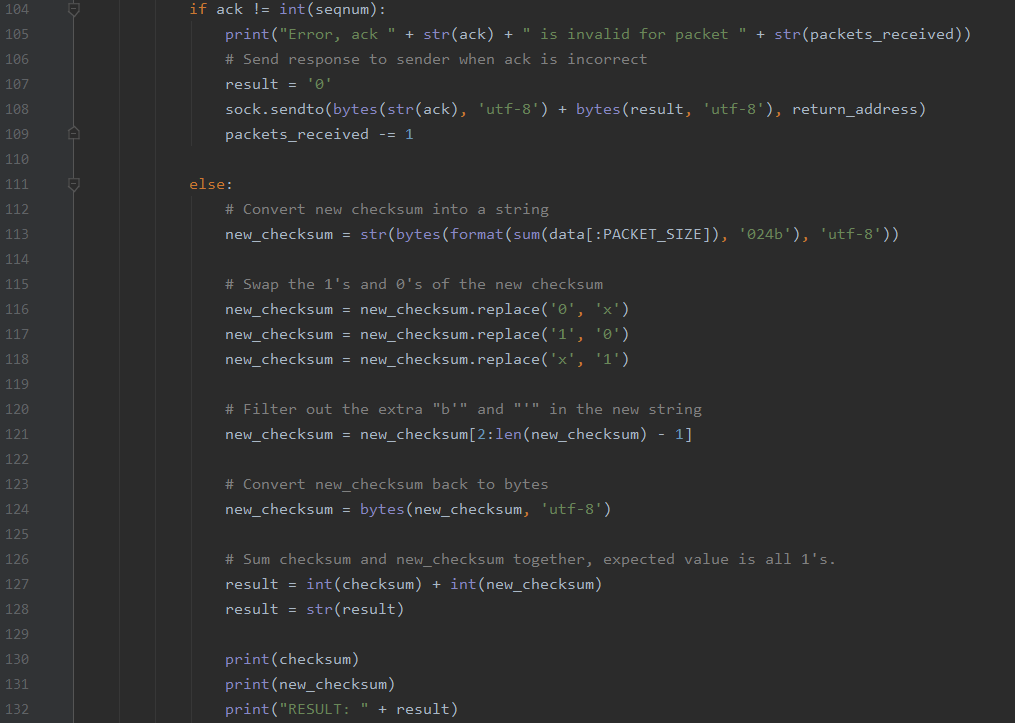
Line 40 begins processing the checksum and ack from the receiver side. Line 48 begins the process to verify the checksum and ACK. If ACK is invalid from packet, data is resent. If checksum is invalid, data is resent.



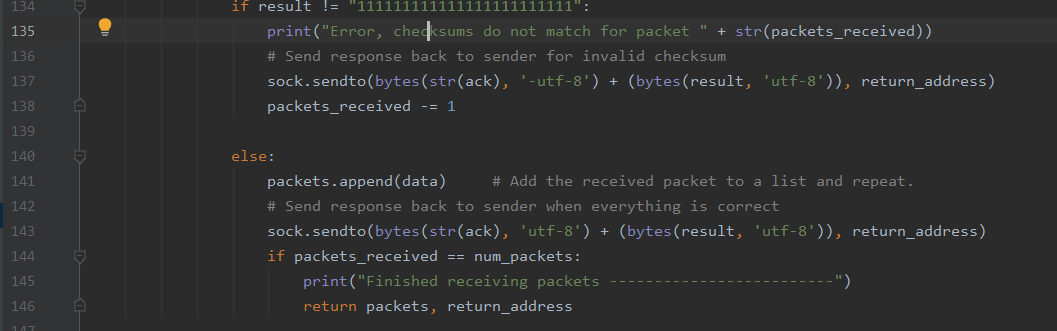
Line 70 of the parse\_packet separates the data, checksum and sequence number from the raw data string.



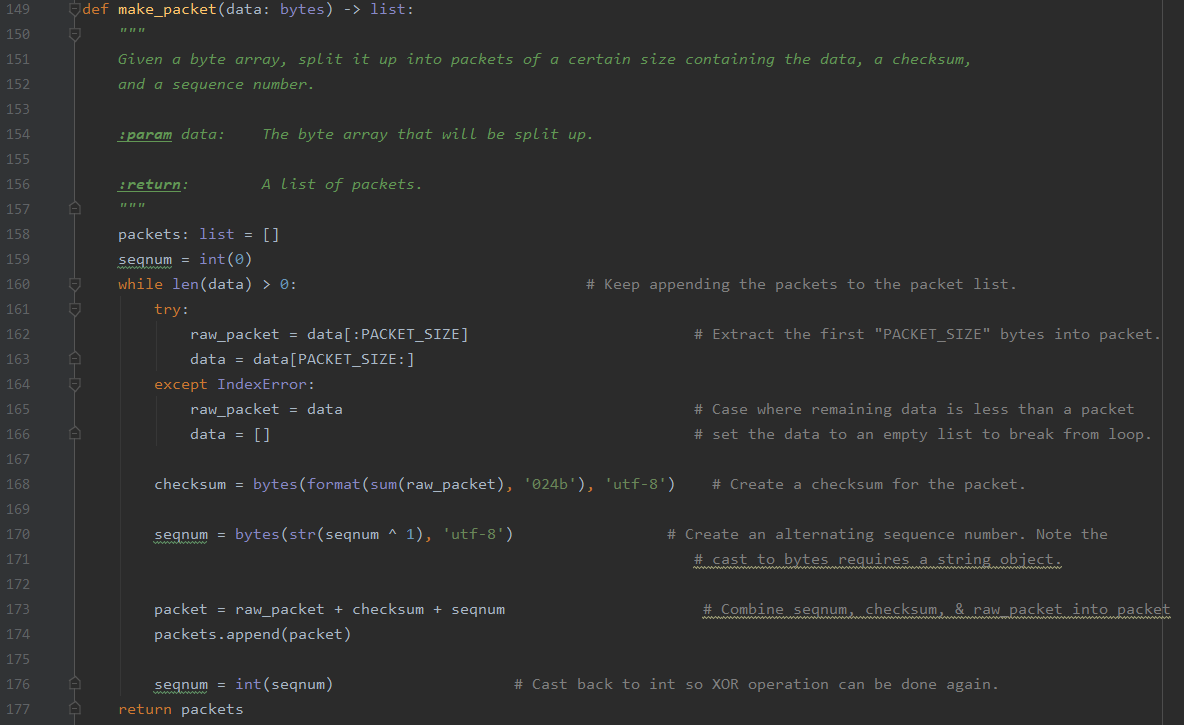
In receive\_packets function, packets are received in. The INITIALIZE sequence is used to tell the receiver how many packets to expect, this gets stored in num\_packets. The receiver then starts processing the packets as it receives them.



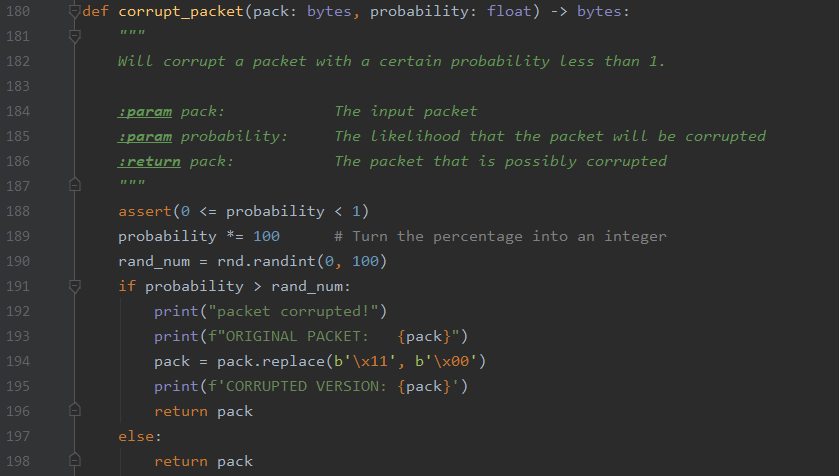
If the ACK does not equal sequence number, then let the sender know the ack is invalid. If ACK is valid, it changes new checksum into a string in line 113, this is to make it easy to swap the 1’s and 0’s of the bytes. It then gets converted back to bytes. After adding the sender;s checksum and new checksum, the result is expected to contain all 1’s. Result is printed.



If the result does not equal all 1’s, an error occurs a message must be sent to sender for invalid checksum. On failures, packets\_received is decremented and no packet is appended to the list. If result goes well, packet received is added to a list repeatedly. Once all is good, message is sent to sender.

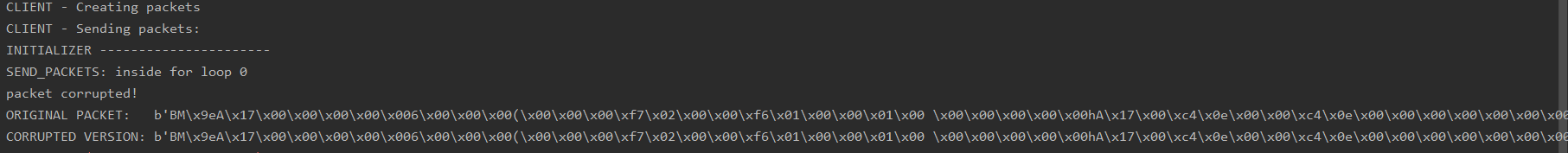


Make\_packet function keeps adding packets to the list. The first PACKET\_SIZE bytes are taken into packet except when data left is less than one packet. Line 168 creates a checksum for the packet. An alternating sequence number is created and then added to raw\_packet and checksum to make the packet. Packets are then added to the list.



This function corrupts the packet. It does so by taking a random decimal between 0 and 1, and compares it to the float parameter the user provides.

*Client Results:*



*Server Results:*

*../Screen%20Shot%202019-11-03%20at%2010.53.45%20PM.png*

*image:*

**

This is the final image received when running the test file without modifying it.