Dylan Schneider, Roark Myers, Josias Polonia

November 3, 2019

Network Design

Vokkarane

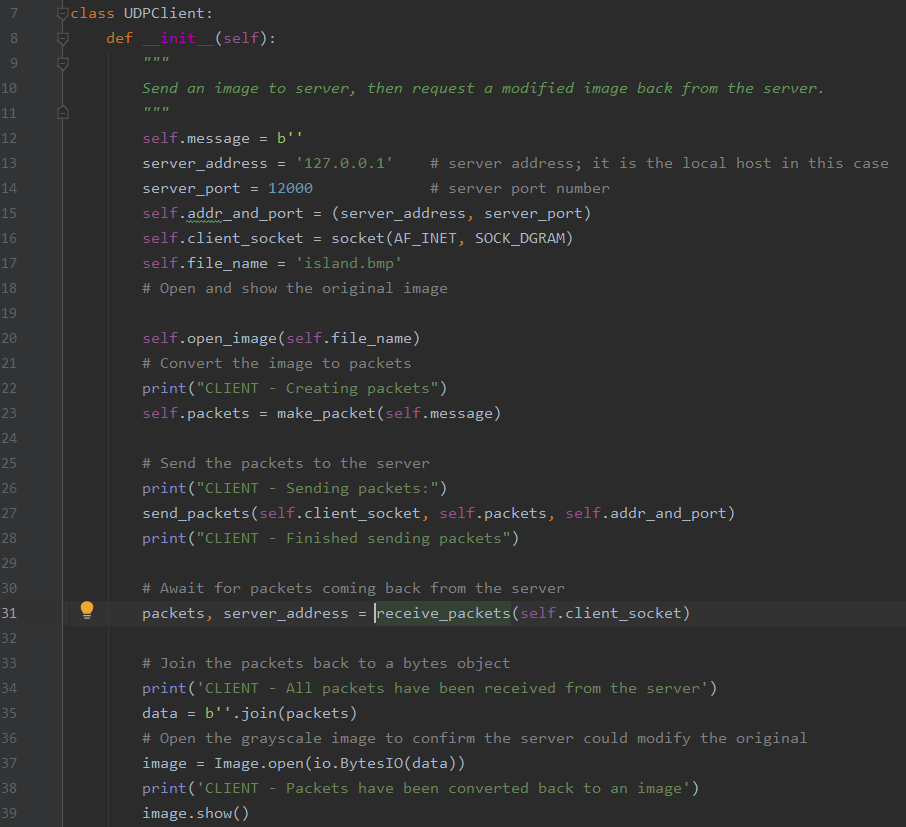
Design Document: Phase 3

**Purpose of the Phase:**

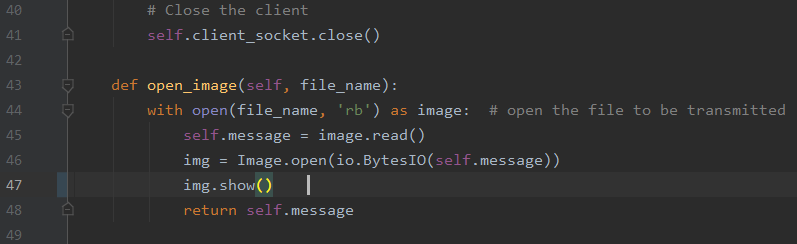
The purpose of the third phase is to implement the RDT 2.2 method (reliable data transfer) which now allows bit errors and error detection; now positive acknowledgements are used to notify the sender whether or not the sent packet had any errors and it received it. Not only must the receiver clearly 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:**

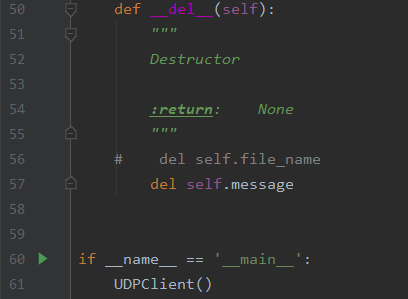
*UDP Client:*



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 edits. 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; the packets are then sent to the server in line 27 until completion. After the server performs some modifications to the packets, they are received back and stored/converted back to bytes(line 31/35). Line 37 is just used to verify that the server altered the original file sent by displaying it.

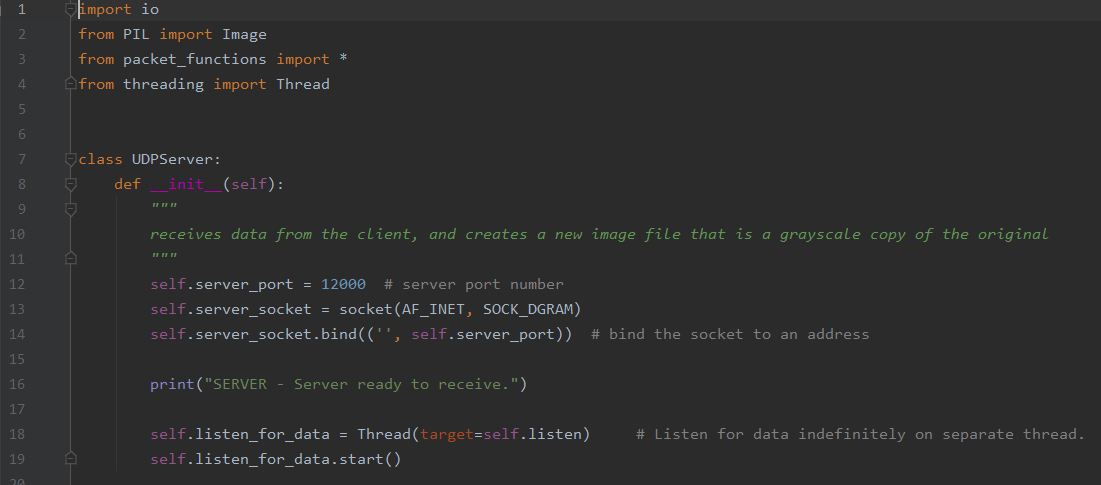


Now we look at the open image function which was called within the last definition. Line 44 simply opens the file to be sent by reading it into self.message, and opening the image into bytes.

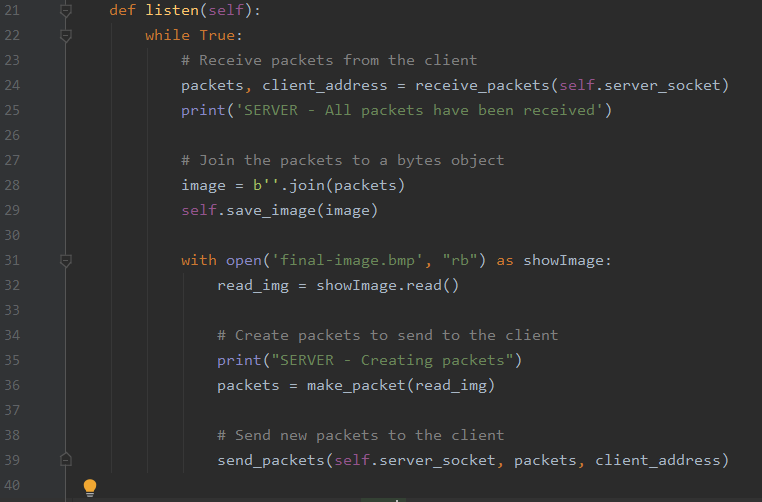


This just declares the destructor which then gets called once all references have been deleted.

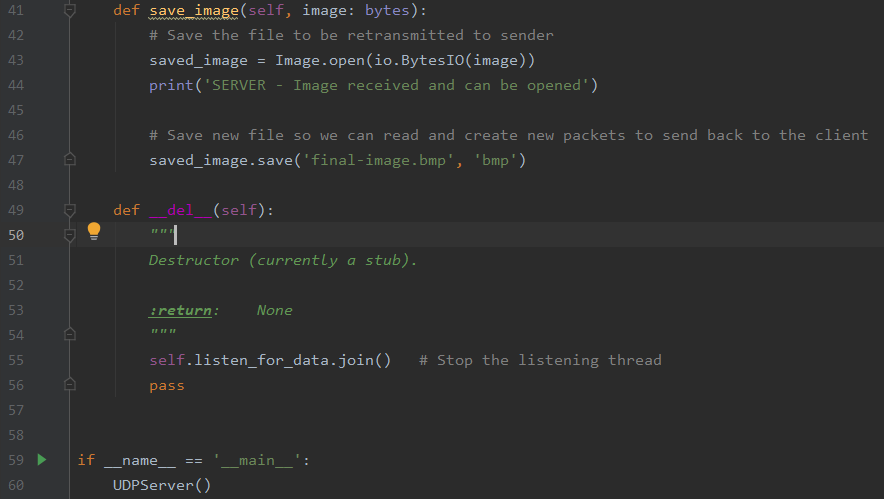
*UDP Server:*



The Server states its port number and its socket; line 14 attaches the socket to the port number. On a separate thread, the data is listened for a particular amount of time.

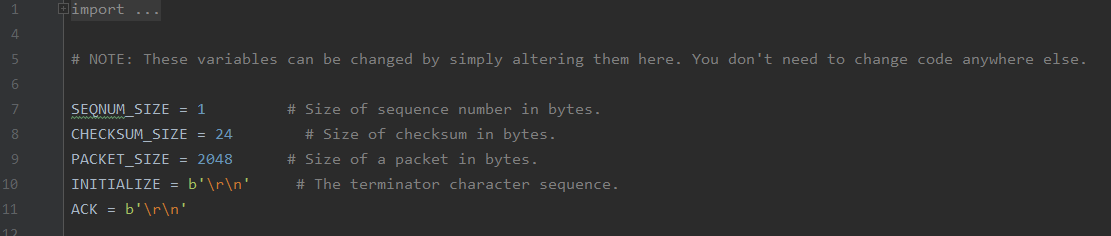


The server receives packets sent from the client and stores them until all have been sent. They are combined into bytes to create a full image. The read in image is then displayed. Line 36 begins to make packets this time to send back to the client as before. Self.save\_image is a function later to be discussed. Line 39 uses the socket, created packets and receiver address to send the packets over.

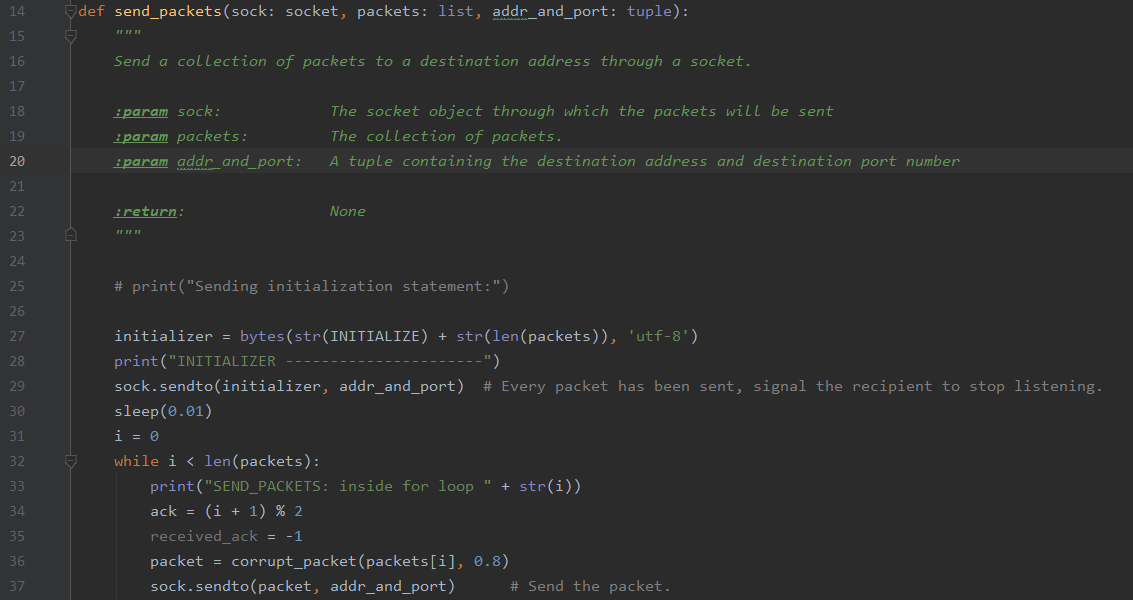


The save\_image function opens up the received image and stores it into bytes; line 47 then saves the image to be read and prepare to create new packets and send them back to the receiver once more. Line 55 goes back to the listen function to stop the listening process.

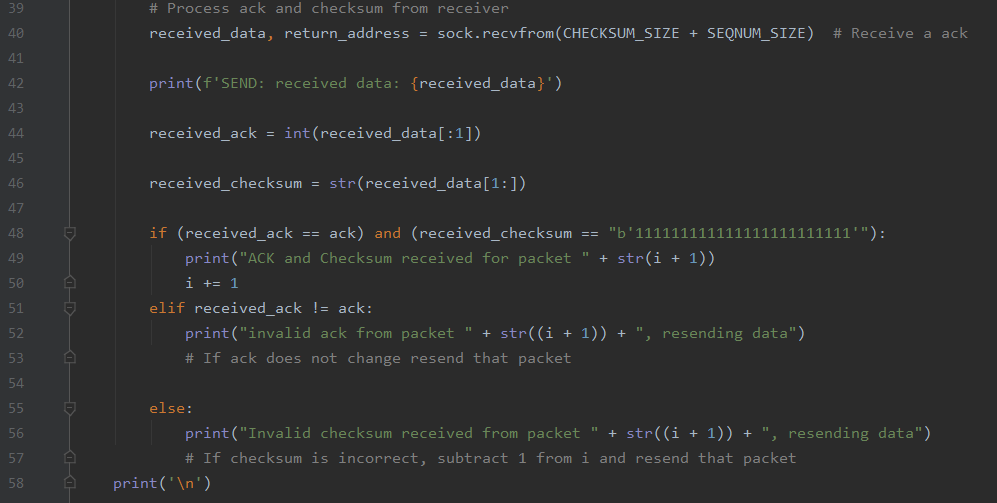
*Packet\_Functions*



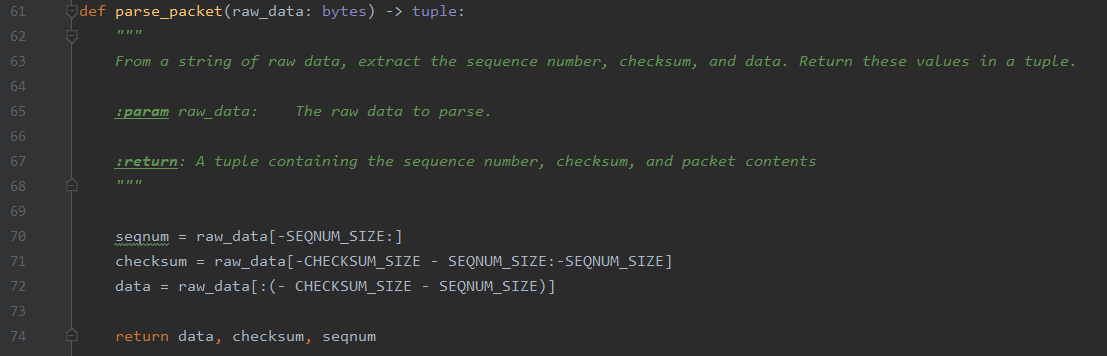
This just states the sizes and terminator character sequence.



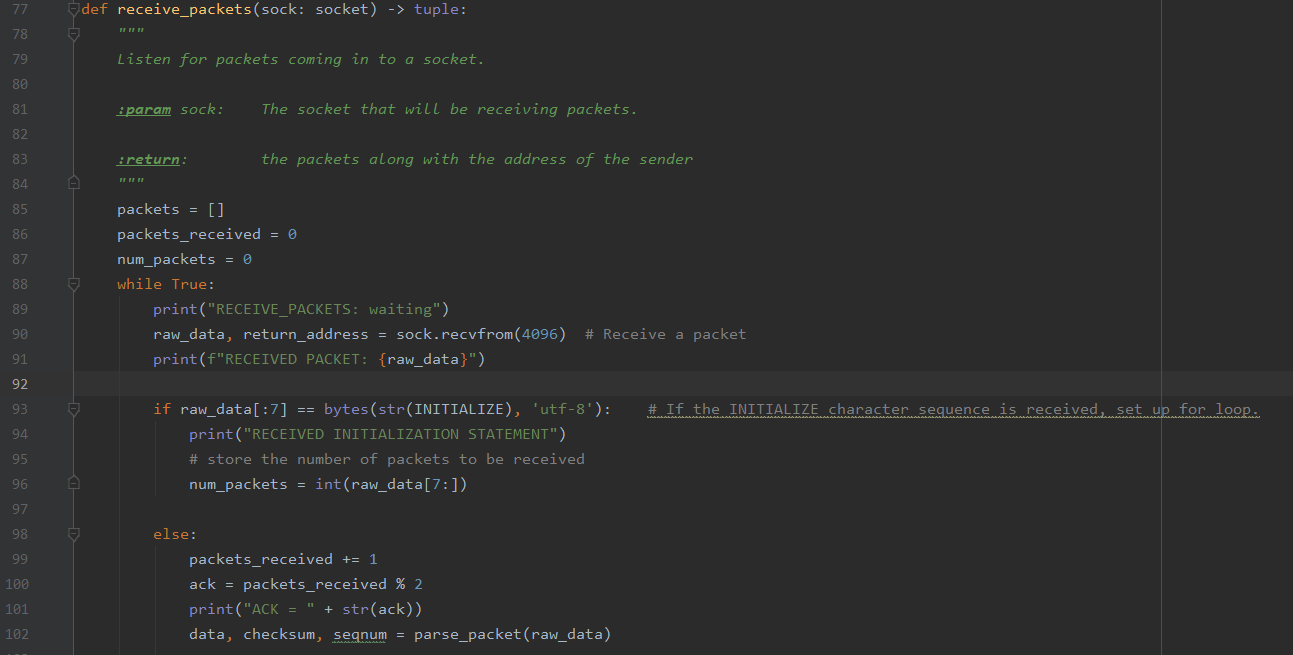
The send\_packets function, through a socket, sends a bunch of packets to desired address. Line 27 creates an initialization and then sends all the packets; receiver stops listening. After the ack declaration, received\_ack, packet is set equal to corrupt\_packet().



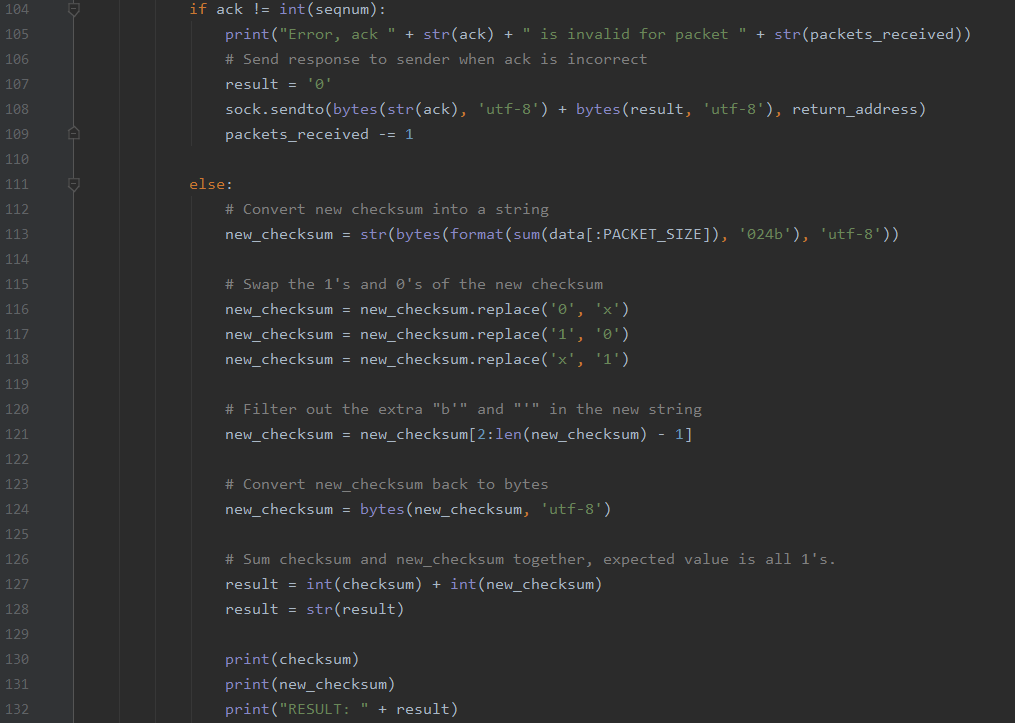
Line 40 begins processing the checksum and ack from the receiver side. Line 48 begins the process to verify the checksum and ACK. If ACK is invalid from packet, data is resent. If checksum is invalid, data is also resent.



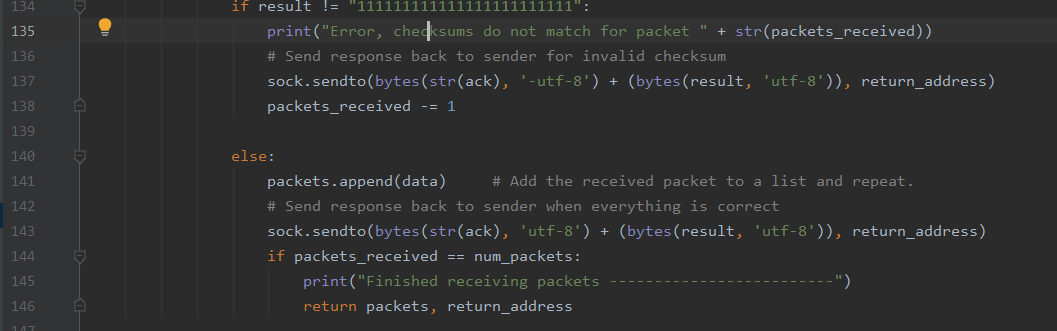
Line 70 of the parse\_packet takes out the sequence number from the raw data string. Then the check sum and data are taken out; data goes to parse. Function returns the data, checksum, and sequence number.



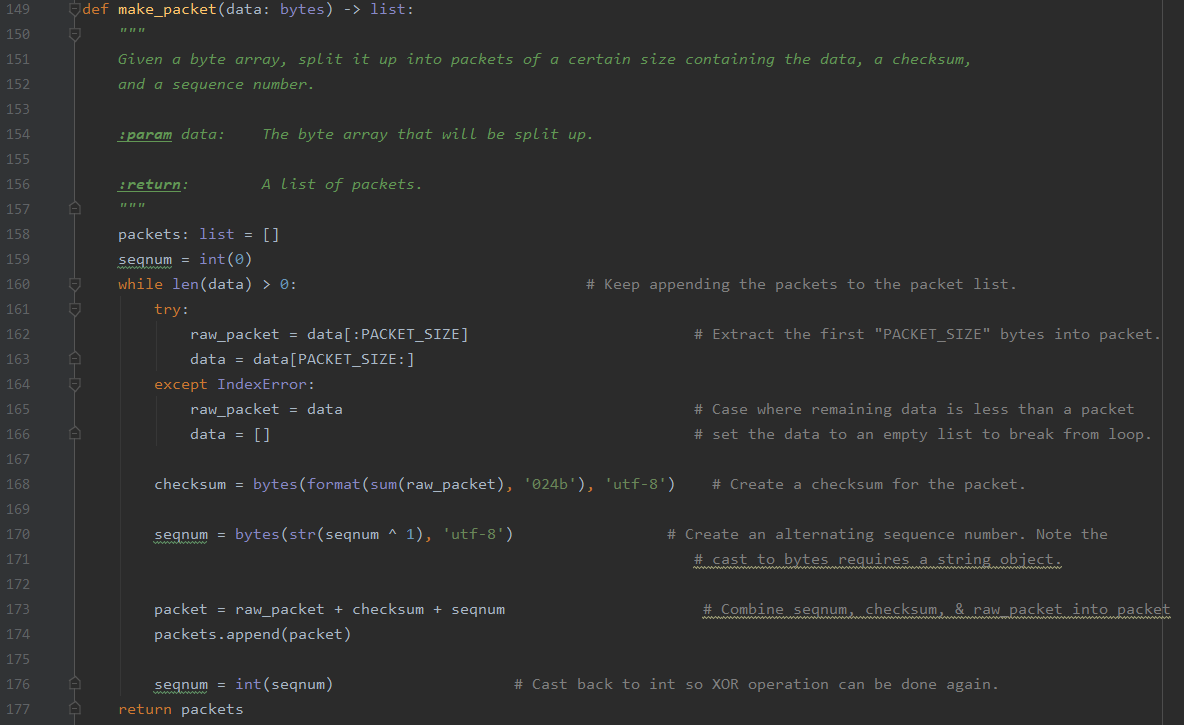
In receive\_packets function, packets are received in. If INITIALIZE character sequence a loop needs to be set up; also the amount of packets are stored into num\_packets in that case.



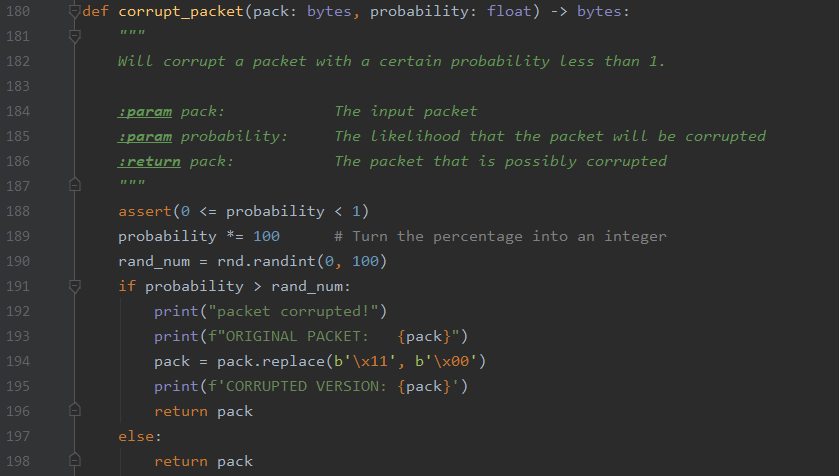
If the ACK does not equal sequence number, then let the sender know the ack is invalid. If ACK is valid, it changes new checksum into a string in line 113. Later on, it must be converted back to bytes. After adding the sum checksum and new checksum, the result is expected to contain all 1’s. Result is printed.



If the result does not equal all 1’s, an error occurs and a message must be sent to sender for invalid checksum. If result goes well, packet received is added to a list repeatedly. Once all is good, message is sent to sender.

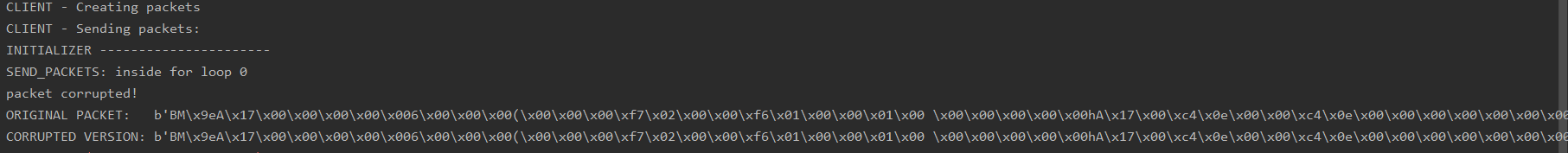


Make\_packet function keeps adding packets to the list. The first PACKET\_SIZE bytes are taken into packet except when data left is less than one packet. Line 168 creates a checksum for the packet. An alternating sequence number is too created and then added to raw\_packet and checksum to make the packet.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



This function has a probability of less than 1 but will corrupt a packet.

*Client Results:*



*Server Results:*

