Roarke Myers, Dylan Schneider, Josias Polonia

November 17, 2019

Network Design

Vokkarane

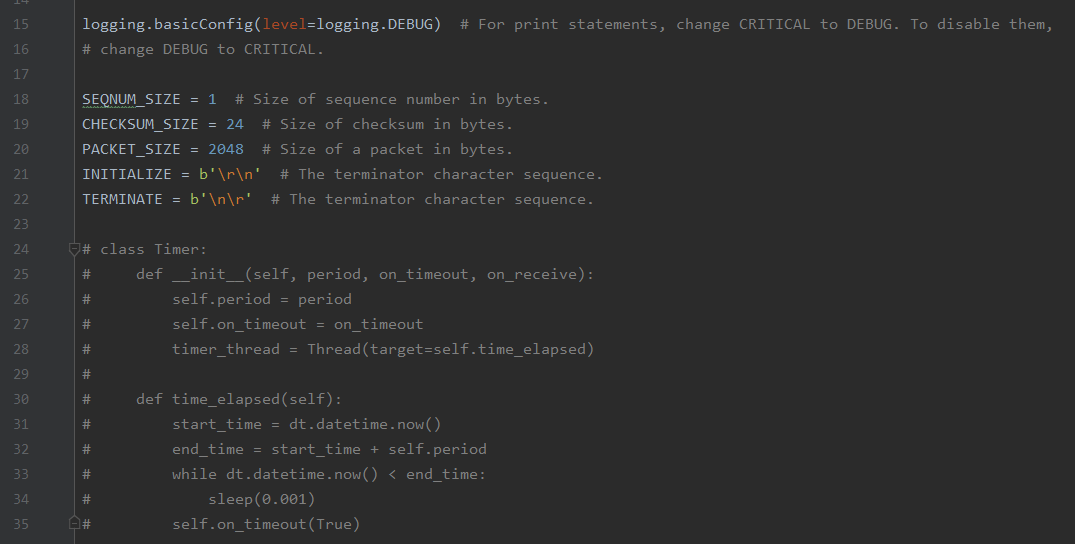
Design Document: Phase 4

**Purpose of the Phase:**

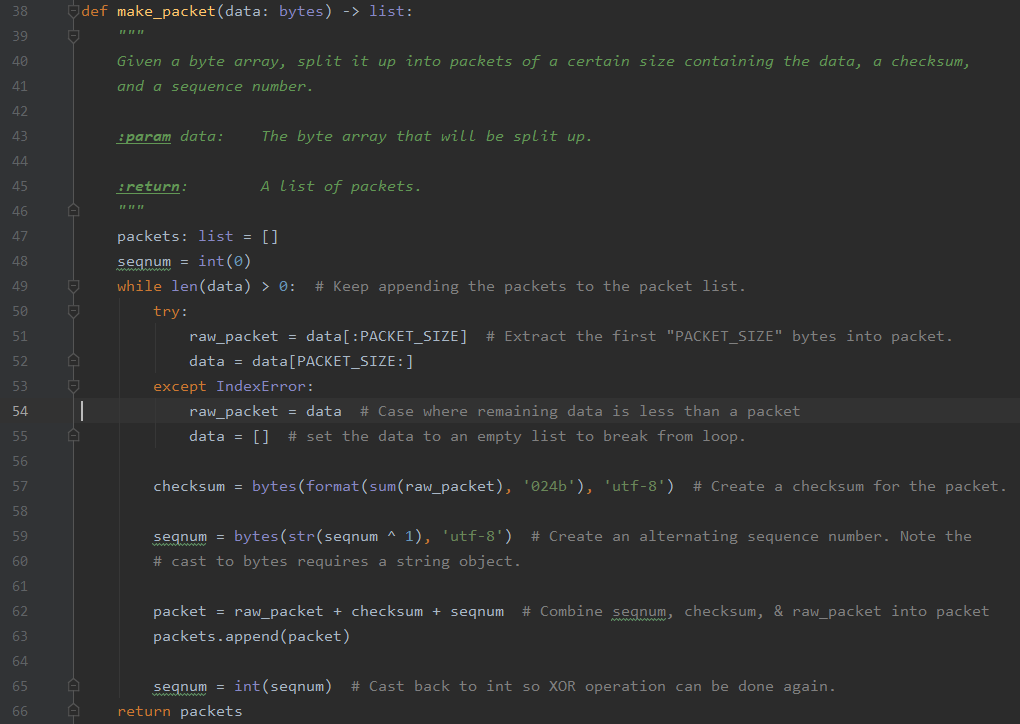
Phase 4’s purpose is to implement a more enhanced version of Phase 3. This time we must implement RDT 3.0 with bit-errors and losses over an unreliable UDP. So it must be able to withstand data packet and ACK packet loss. The image is broken into packets and sent through the UDP sockets but to handle packet loss must use a countdown timer; receiver needs to put the packets received in order.

**Code Explanation:**

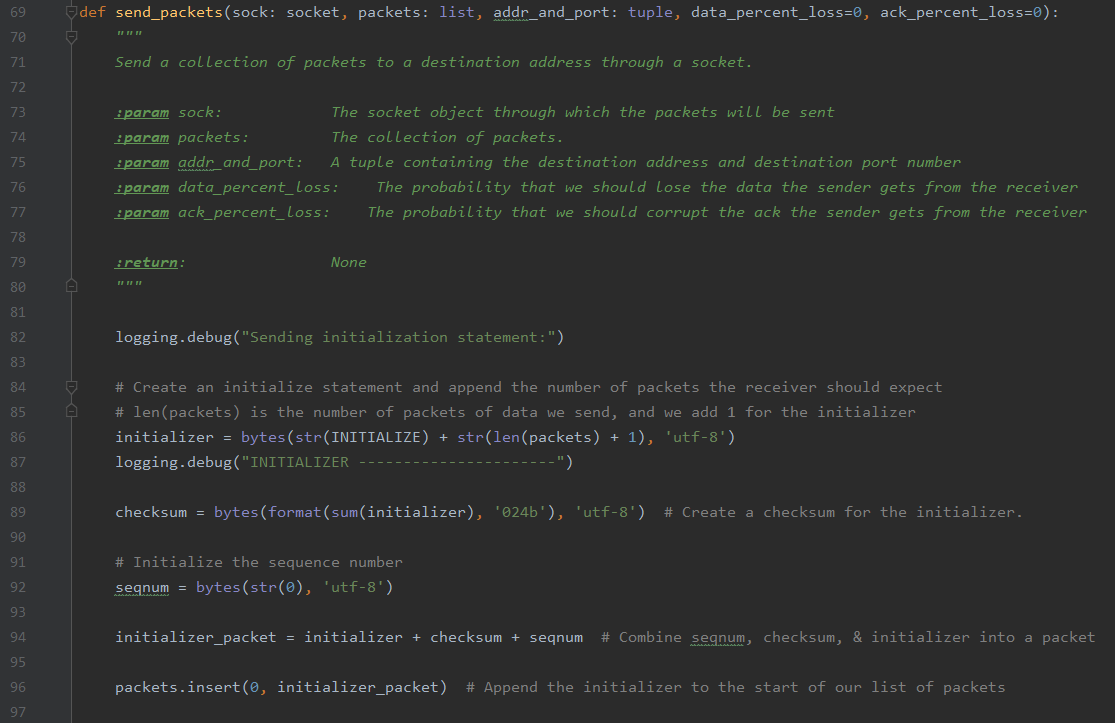
Packet\_functions.py



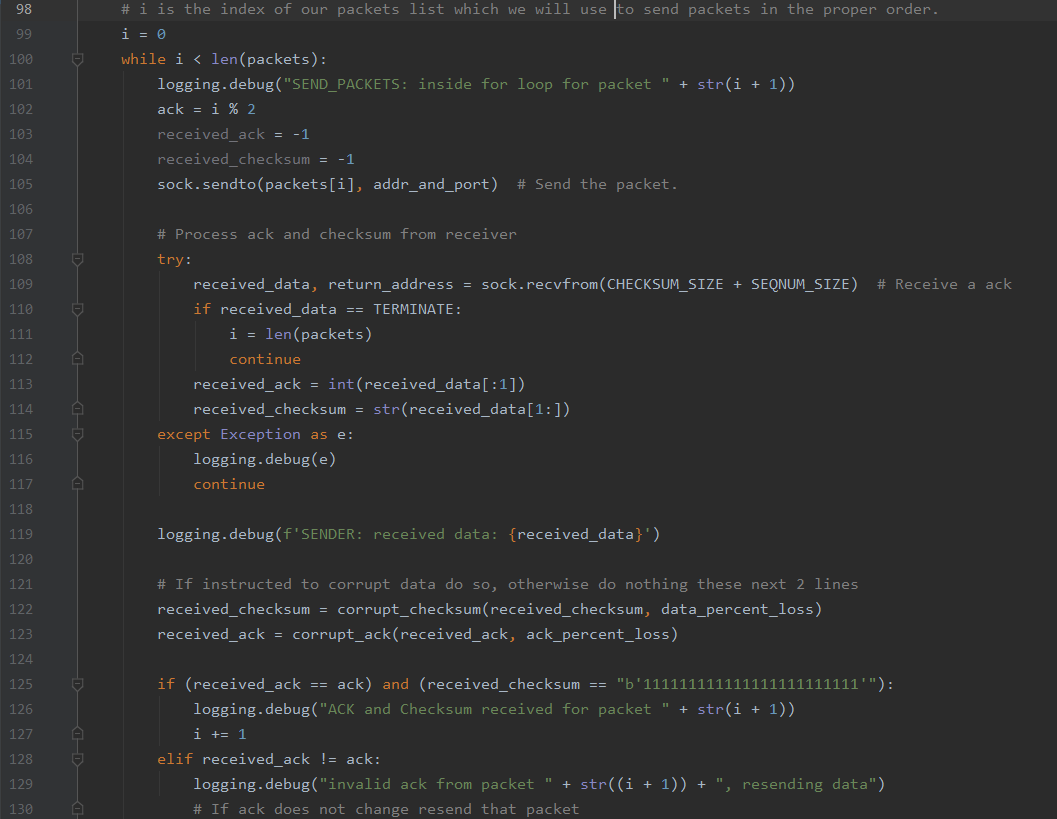
Lines 18-22 simply initialize the sizes and terminator characters.



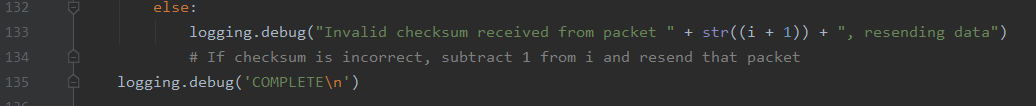
The make\_packet function definition is given an array of bytes and splits them up into packets containing their checksum, data, and sequence number. Until all the data is read in, the packets are added to the packet list. First bytes are extracted into packet; line 53 shows that if the data leftover from the making process is less than a packet then its set to an empty list separate from the loop. Line 57 makes a checksum for the packet using the data from the while loop above. An alternating sequence number is created; finally line 62 adds the raw packet sequence number and checksum to the packet. Before returning packets, the sequence number is cast back into int so the XOR process can be repeated.



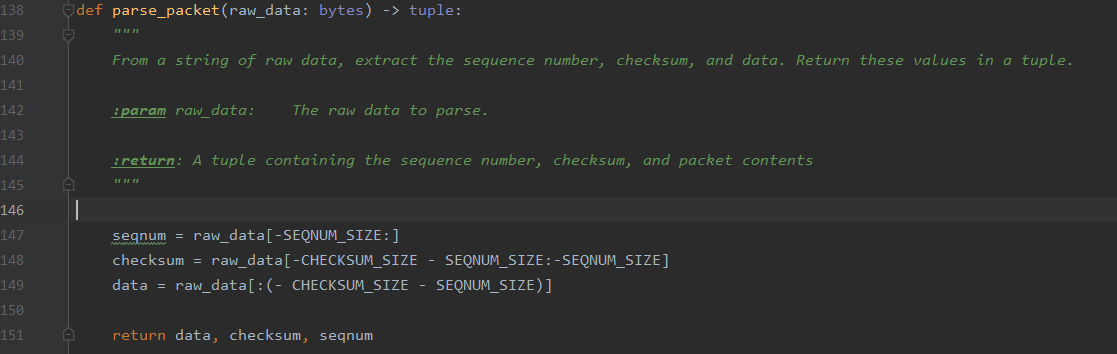
The send\_packets function essentially sends the group of packets to the receiver through a socket. Its parameters are the socket for the data to be sent through, the actual packets, addresses and destination numbers, and the data/ACK % loss. Line 89 creates a checksum for the initialization then the sequence number is initialized. Those two are combined with the initializer into a packet. Line 96 adds the initializer to the beginning of the packet list.



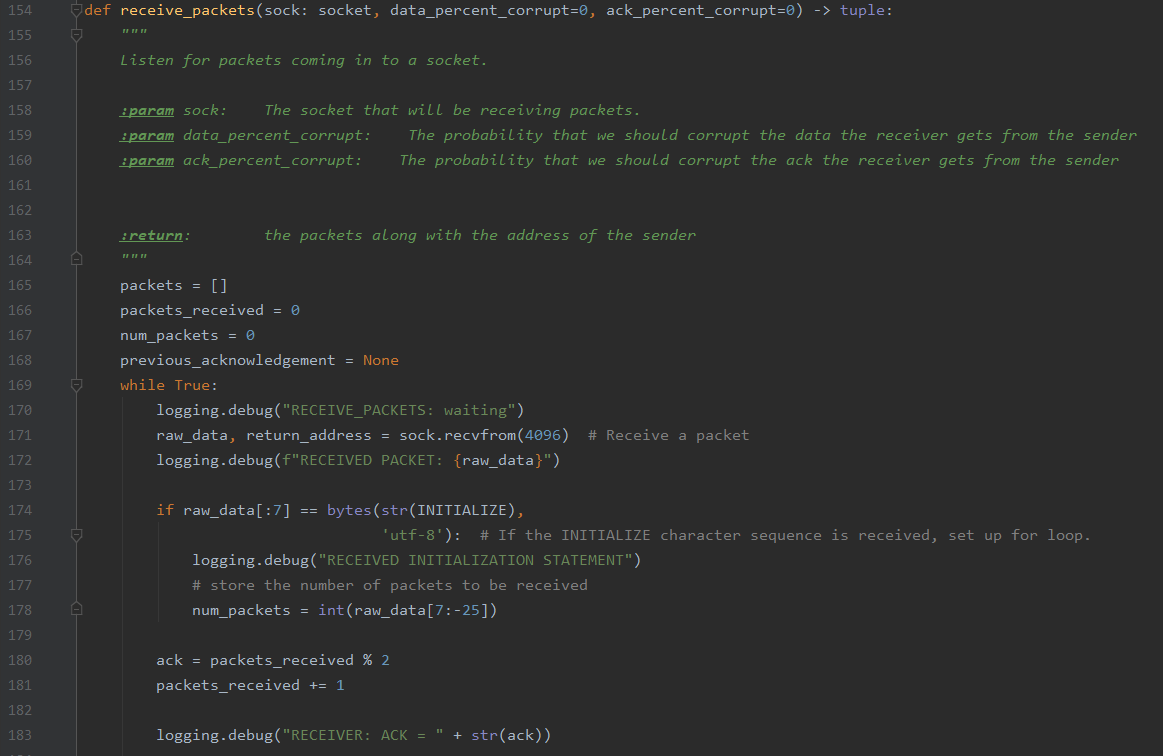
While we still have packets to send, packets are sent as the received ack and checksum are initialized. Line 109 takes in the ACK; if the data received is terminating character, I is set equal to the length of the packets. The received ACK and checksum are set equal to the received data integer/strings. Line 122 starts the corruption process with the received data; ACK is also corrupted. If both are equal to the “b’111111111111111111111111’” statement is prtined saying that ACK and Checksum are received for the string number for each packet. If invalid ACK, an error statement is outputted and data is resent.



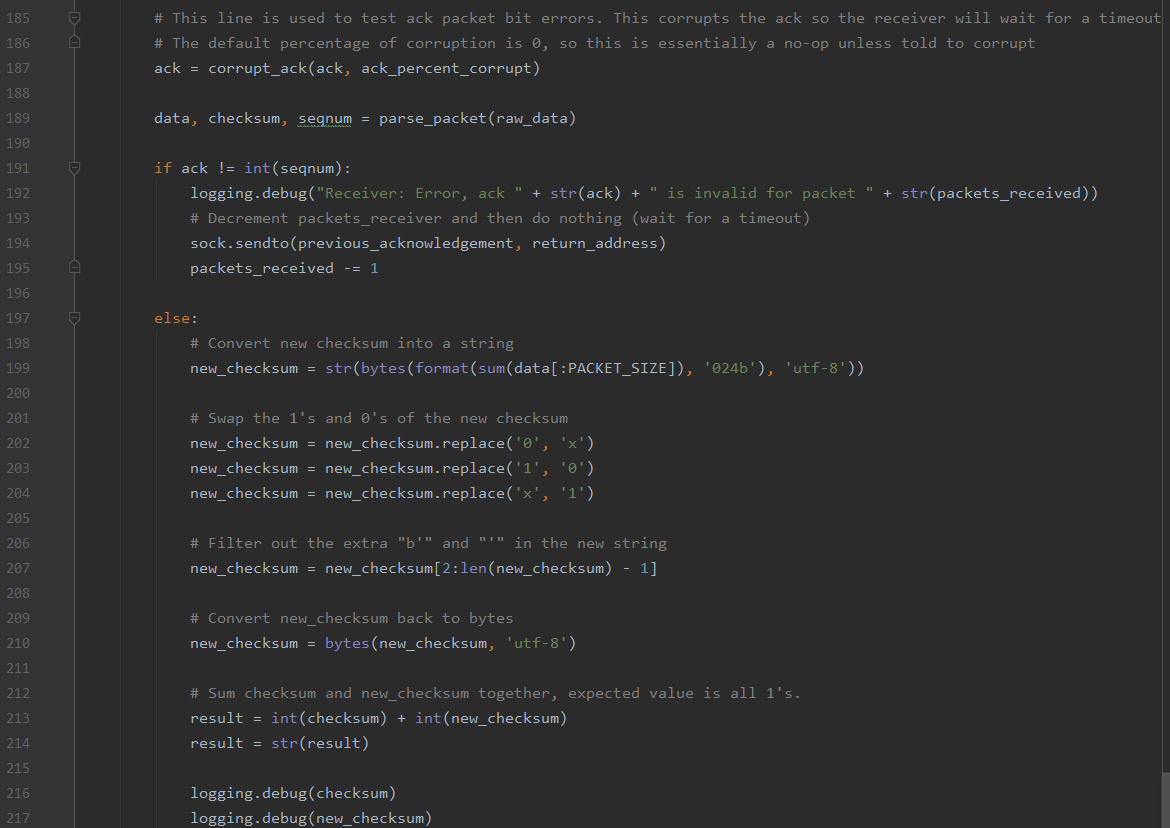
The code above is for an invalid checksum received, data is resent.



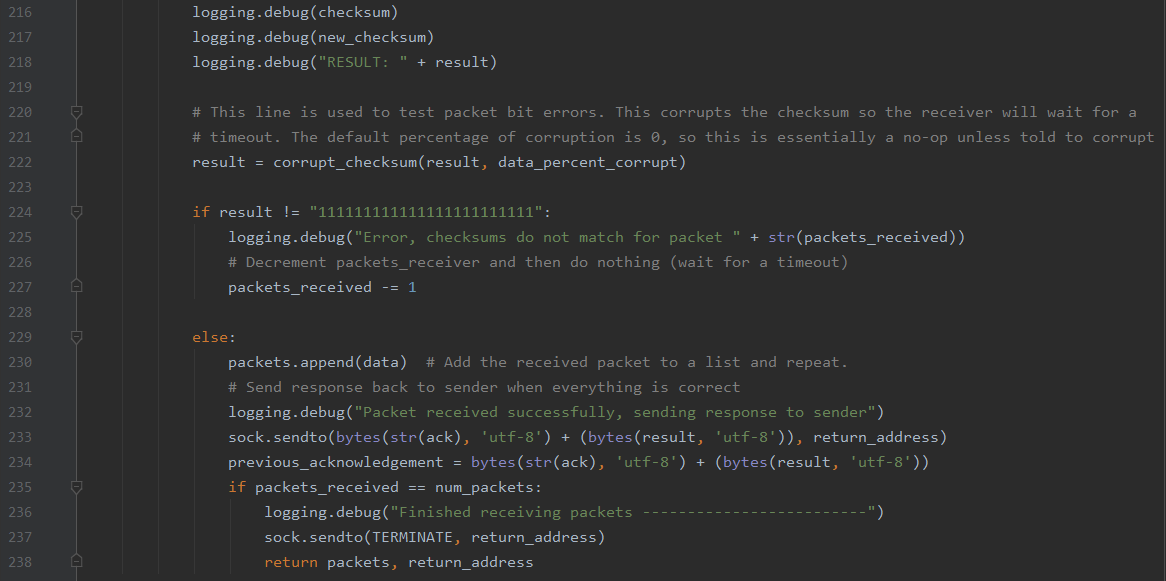
With the string of data, the checksum, data, and sequence number are extracted and returned in a tuple later. Only parameter is the data.



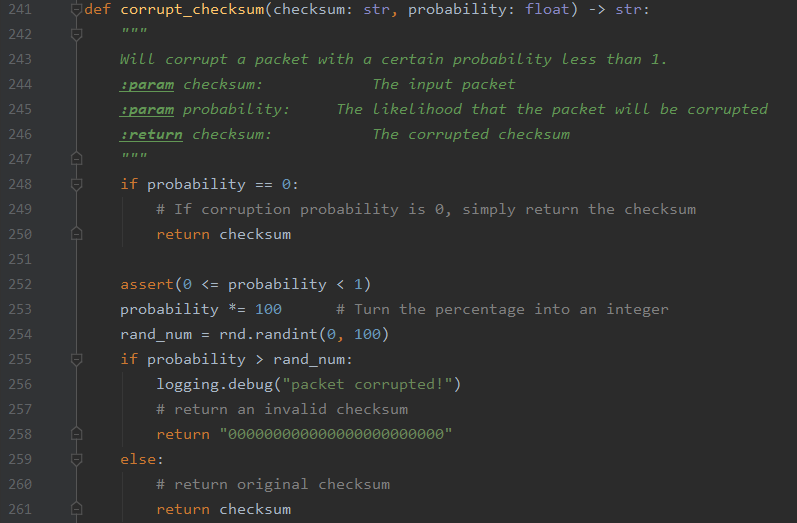
The receive\_packets function overall just listens for packets; takes in the socket, and data/ACL corrupts. They begin with 0 packets. While there are still packets to read in, packet is received. Number of packets to be read in are stored in line 178. The packets are also read in order of what was sent.



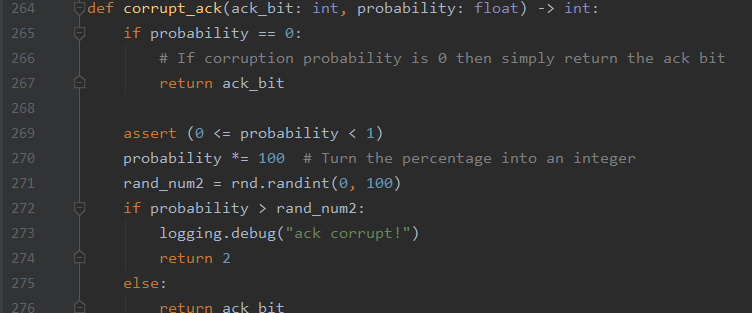
Line 187 tests the ACK packet bit error by corrupting the ack so the receiver waits for a timeout. If ack is not equal to sequence number, error is printed and packets is decremented then wait fir timeout. If not, new checksum is converted to string. New checksum switches 1’s and 0’s. New checksum is converted back to bytes in line 210. After that, the result is the sum of the checksum and the new checksum.



Line 224 is used to test for bit errors by corrupting the checksum thus causing the reciver to wait for a timeout. If result is not as stated, checksums do not match for packet, error is printed and packets received is decremented. If result is good, received packet is added to the list and process is repeated; then send a response to the sender acknowledging that the data is ok. Once the packets are all read in, the socket is ended.

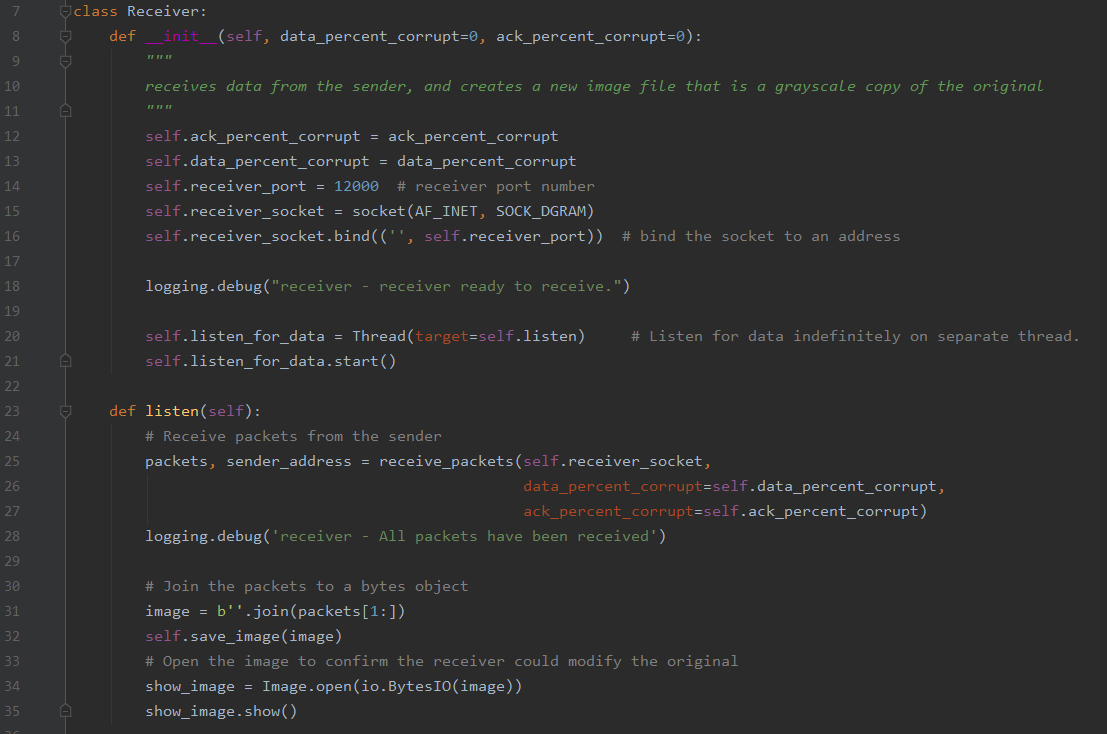


The corrupt function just corrupts a packet with a chance of less than 1. If probability is 0, checksum is returned. Otherwise, probability is turned into an integer and randomly generated between 0 and 100. Packet returns invalid checksum if packet is corrupted, if not return the original checksum.

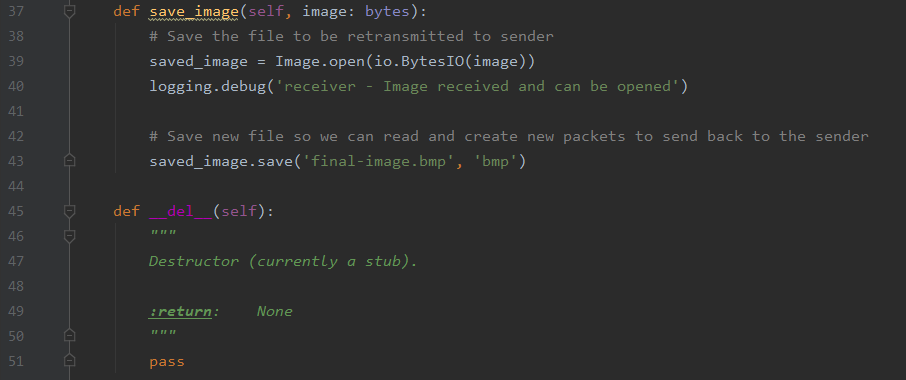


Ack corrupt function does similar with probability equal to 0; ack bit is returned. Probability is treated the same as before and if probability is greater than randomly generated number than corrupt ack is returned. If not, original ack is returned.

**Reciever.py**

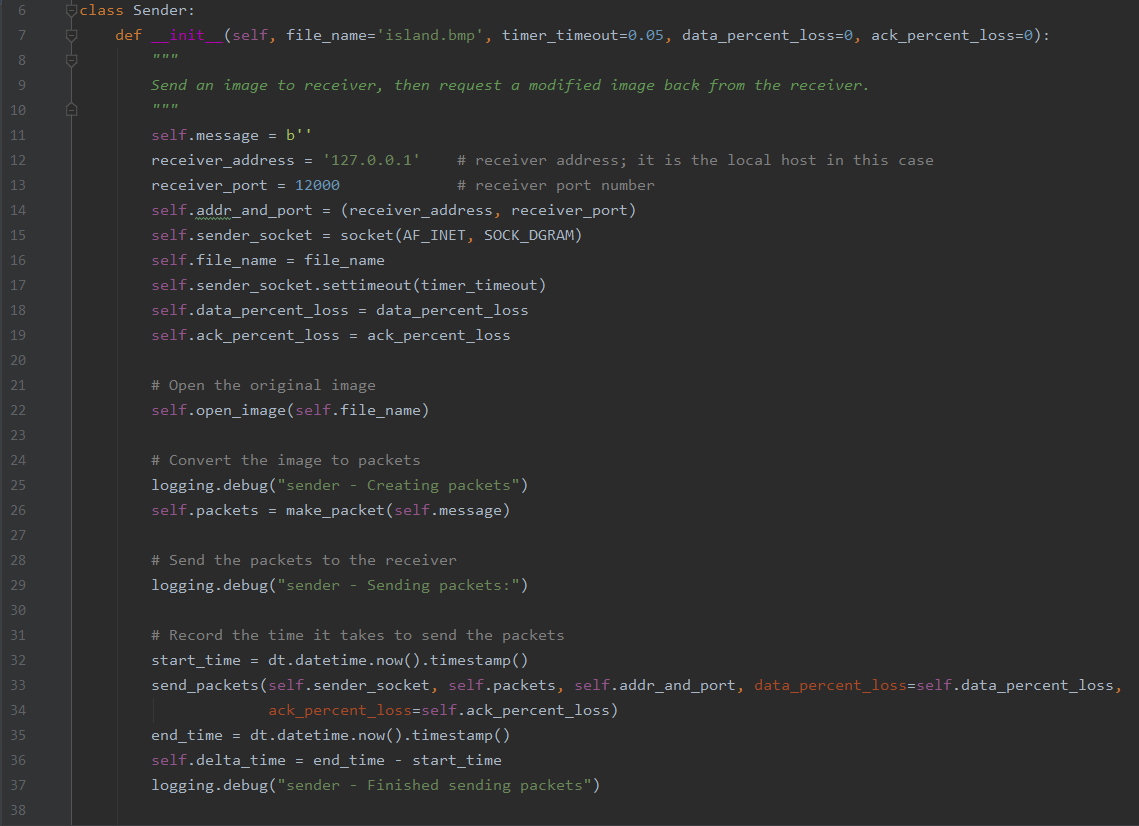


Receiver class begins with def\_init\_() that receives in the data from the sender and makes a new image file. Ack and data percentages are initialized as well as the receiver port. The receiver address is connected to the socket to prepare for sending. Receiver begins to listen for data. Def\_listen receives in packets from the sender. Once all the packets have been received, packets are joined into a bytes object to make the image once again. Image is opened to verify.

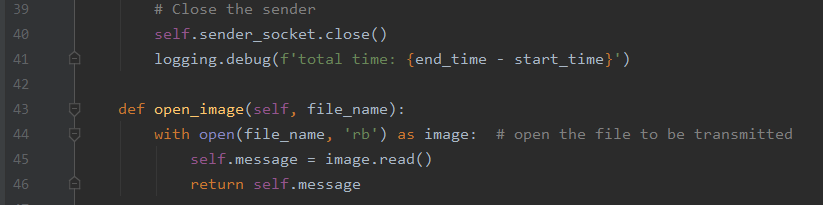


Definition is to save the image received from the sender. Once saved, it can be converted back to new packets to return to sender. Last function is simply the destructor function that must be called.

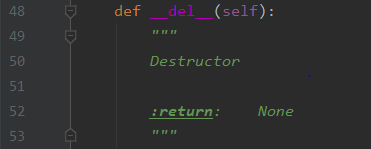
**Sender.py**



In the sender class, it begins with a definition that sends an image to the receiver then takes back in the image sent as the receiver sends it back. Island.bmp is the image name. Receiver address and port are set and a socket is aligned to prepare the sending. Original image is opened and then converted to packets. Line 28 begins sending the packets to the receiver. Timer begins in line 32 for the times to send all the packets. Once all are sent, the sender is closed and the time is stated.

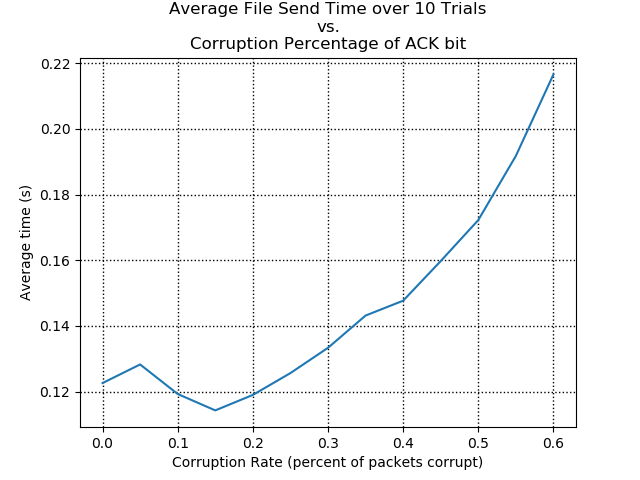


File is opened to prepare for sending and read into message.

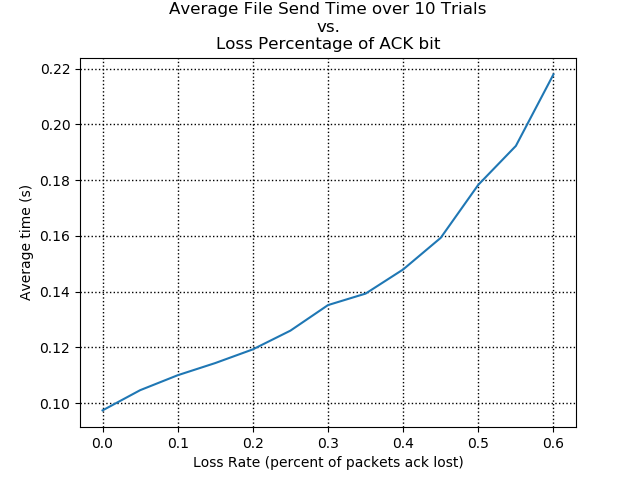


Destructor function.

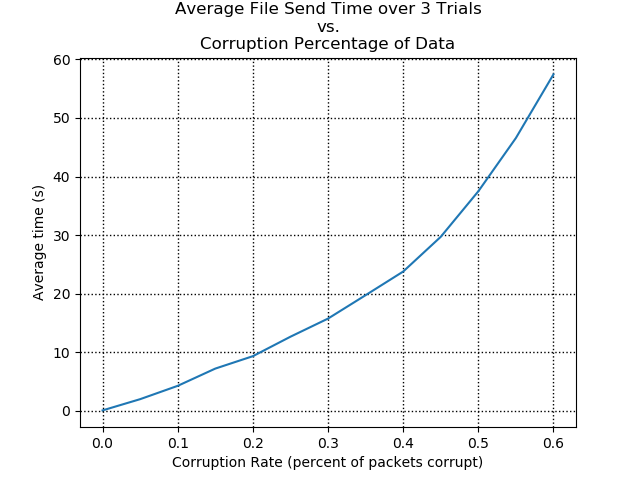
*Ack Corruption Graph*



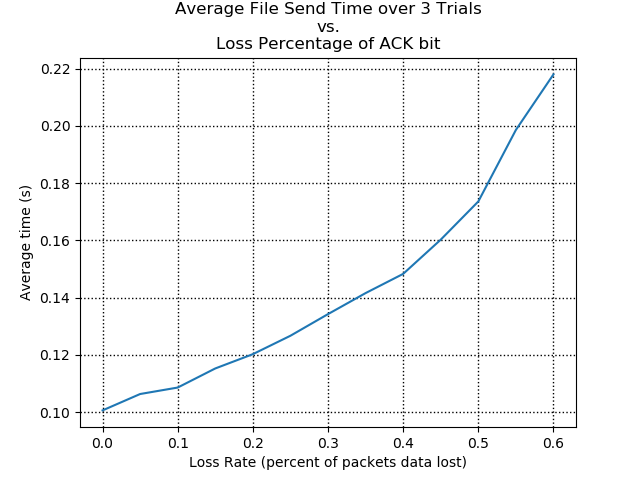
*Ack Loss Graph*



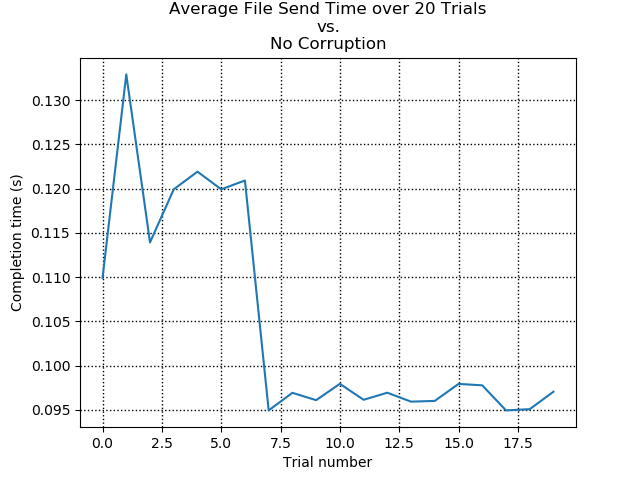
*Data Corruption Graph*



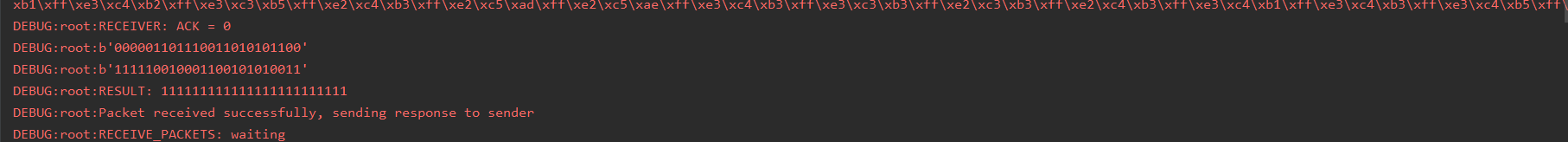
*Data Lost Graph*

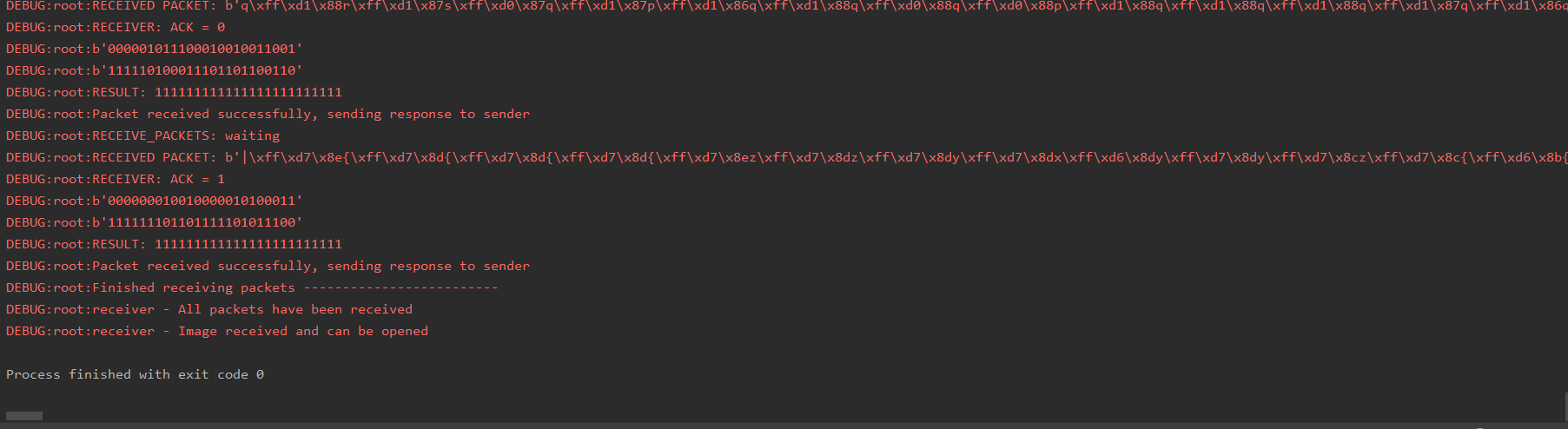


*No Corruption Graph*

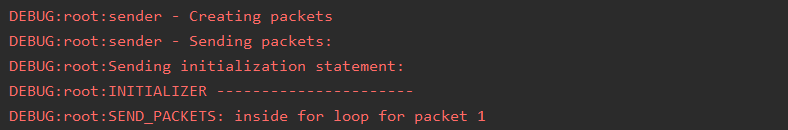


*Code Results:*

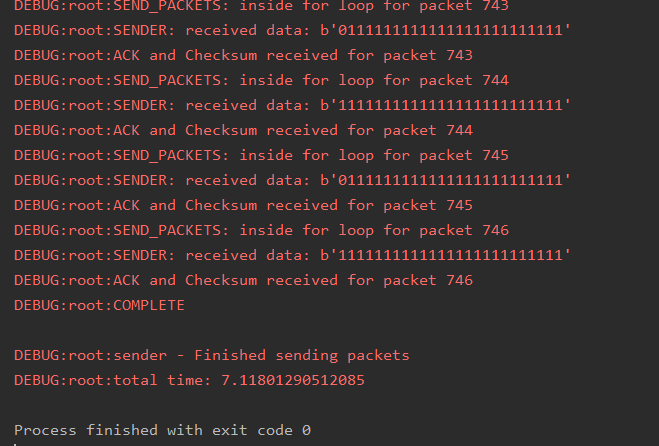
*Beginning of receiver code snip.*



*Code snip of receiver side.*



*Beginning of sender code snip.*



*Code snip of sender side.*