Ryan White, Keegan Chhay, Renin Kingsly-Jose

Network Design Project: Phase 4

RDT 3.0

Design Documentation

**sender.py**

Text

Description automatically generated

Figure 1: Sender Initialization

This initial segment of code on the sender side initializes the extra credit GUI and the widgets it contains. It also assigns the start time of the script in order to accurately measure how long it takes to transmit the file.

Text

Description automatically generated

Figure 2: Checksum and Packet Structure

This passage of code contains two functions: the checksum to verify valid packet data, and the MakePkt() function, which compresses checksum value for a given packet, sequence number, and raw data of a packet into one structure to pass through to the receiver.

Text

Description automatically generated

Figure 3: MakePayloads()

The MakePayloads() function is the bulk of sender.py, and it serves multiple purposes. The first purpose is to read a JPEG file and slice its data into packets of one kilobyte each, maximum. Then, it calls the previously documented MakePkt() function for passage to the receiver. This function also handles the events in which data must be resent, contingent upon the ACK value received from the receiver: if the ACK value was corrupted, or if a positive ACK was not received, the sender will resend the previously sent packet. Finally, the last segment updates the progress bar widget contained in the GUI, and the ending if statement displays to the user that the image upload is complete, and outputs how long it took to transmit the image.

Text

Description automatically generated

Figure 4: Socket Configuration

This segment configures the socket for communication with the receiver. It also launches the GUI, and finally closes the socket.



Figure 5: Timeout Configuration

This line is what differentiates RDT 2.2 from RDT 3.0: a timeout adjustable on machine speed. For our purposes, this constant of 1.5 milliseconds worked fine.

**receiver.py**

Text

Description automatically generated

Figure 6: Raw Data Corruption

The receiver file starts with a function to corrupt a packet’s raw data. Depending on the user’s inputted percentage and the system’s random input, a given packet passed to this function will be chosen to be corrupted. In this event, the corrupted packet will cause the receiver to not send a positive ACK to the sender, which prompts the sender in turn to resend the corrupted packet until it is correctly received.

A screenshot of a computer

Description automatically generated with medium confidence

Figure 7: ACK Corruption

In our program, the ACK consists of 8 bits. A positive ACK consists of ‘00000000’, and a negative ACK consists of ‘11111111’. In the event of ACK corruption, these bits are randomly flipped, and the sender will resend the packet in which the ACK was corrupted.

Text

Description automatically generated

Figure 8: Packet Dropping

Options 4 and 5 revolve around these two functions: depending on the percentage the user inputs, the program will randomly sleep, dropping a packet and prompting the sender to resend the data.

Text

Description automatically generated

Figure 9: Checksum and Socket Initalization

This segment contains the receiver’s checksum function and the initialization of the server’s socket. It also contains the functionality to take inputs from the command line.

Text

Description automatically generated

Figure 10: Data Reception Loop

This while loop reads the incoming data from the sender and unpacks it. Then, depending on what option the user selected at the command line (ACK corruption or data corruption), and if statement may be entered. In the event of ACK corruption, the ACK may be randomly corrupted contingent upon the user’s entered percentage. If option 3 was selected, the script performs a checksum test of the data to see if the data is valid. If not, then a non-positive ACK is sent to the sender, prompting another send of that packet. If the checksum is passed, then the data is appended to the final image.

Text

Description automatically generated

Figure 11: Packet Dropping Functionality

These if statements call the previously mentioned packet dropping functions with a user inputted percentage that dictates how many packets are randomly dropped.

A screenshot of a computer

Description automatically generated with medium confidence

Figure 12: Image Finalized

The data appended in the while loop is joined and written to file ‘output.jpg’ for viewing. Then, the socket is closed.

Dropbox Link to Option 4 and 5 Demos:

<https://www.dropbox.com/scl/fo/uc2dkl50qnpjuvabwchi4/h?dl=0&rlkey=zhyaq7uy0xcj68qba15hnlh9y>