**Design Document**

**Network Design – Phase 4**

**Drew Bowler, Jack Flood, Jaki Giggi, Tyler Almachar**

**March 31, 2021**

**Introduction**

Our project works by opening the two python scripts (“phase4\_client.py” and “phase4\_server.py”) in two terminals, opening a socket on the host machine, asking the user for an image, and passing it from the client to the server while checking for data corruption, ACK corruption, and timeouts, which can all be forced to happen serverside for the purpose of testing. The user selects an image, an option for corruption/timeout, and shortly after, the server rconstructs and outputs an image file (“output.jpg”).

**Code Description**

This section will be split into two sections (one for each script).

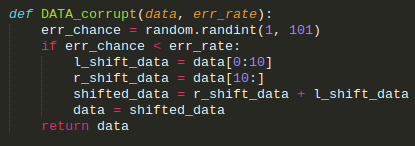
phase4\_client.py

The code prompts the user to input a filename as well as a data corruption option. The file associated with the filename gets split into 1024-byte packets by the read\_file() function, and the option gets sent to the server, which handles data corruption (if selected; if ACK corruption is selected, the client handles it via ACK\_error()). Next, the main data transfer loop begins, and will continue until the last data packet is sent. The loop will first check if this is a “call 0” or “call 1” scenario, which changes upon a successful transfer and confirmation of data. The checksum is calculated via make\_checksum() and then compiled with the packet via make\_pkt() and is sent—the timer begins. If the data is received in time, the recalculated checksum is complete, and the ACK is correct, this phase of the project will end and the next loop will change to the other “call” and attempt to send the next data segment. If any of these checks fail, the program will attempt to resend the same packet until it succeeds. The program will repeat this process and alternate “call 0” and “call 1” until it completes the data transfer, at which point it will terminate.

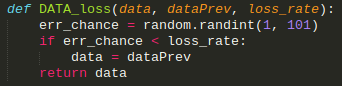
phase4\_server.py

The server accepts the conection confirmation from the client, creates a blank output JPEG to write to, and if data corrupion is requested, generates the percent of packets it will attempt to corrupt. From here, there is a main loop with two individually nested loops that break upon the condition of a successful receive; each loop is associated with waiting for a packet from either a “call 0” or a “call 1.” If the received packet is readable, the checksum can be recalculated, and it is the expected sequence, it will buid the output file with the payload and wait for the next call. If any of these fail, it will wait again to receive it and continue the same loop. In eiher case, the server will only send a confirmation of a successfully received packet. A successful receive and reconstruct will alternate the two nested loops until every packet is successfully received and the image is rebuilt.

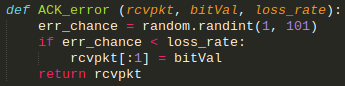
**Execution Screenshots**

****

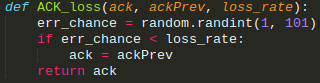
Screenshot 1 (phase4\_server.py): This function represents option 3, data loss. It takes the data loss and an error chance as input parameters. If the chance is less than the randomly generated number between 1 and 100, it scrambles the data (rotates it to the right 10 places).

****

Screenshot 2 (phase4\_server.py): This function represents option 5, the concept of data being lost. It takes the current data being sent, the last data sent, and loss rate as paraneters. If triggered, it replaces the current data being sent with the last data being sent, which signifies to the client that the current data was lost.

****

Screenshot 3 (phase4\_client.py): This function represents option 2, ACK errors. It takes the received packet, an error bit, and the loss rate as arguments. If triggered, it replaces the ACK bit with a ‘2’ (in bytes) which will always return an error (since a 0 or 1 is always expected), representing corruption.

****

Screenshot 4 (phase4\_client.py): This function represents option 4, ACK loss. It takes the received ACK, the previous successful ACK, and the loss rate as arguments. If triggered, it replaces the current ACK with the previous ACK, signifying to the script that the current ACK was lost.

All screenshots also represent option 0 if the error chance is set to 0; it will simply return the data unchanged and never corrupt or lose it.

**Performance Plots**