Alex Fountain

Josh Crum

Tristan VanFossen

**Project 1 Documentation – Python Version**

**Client Side:**

Makes a connection to the server, creates copy of the designated filename, and initializes the packet\_buffer (window array). The count and next\_packet\_received are initialized to 0.

Starting in the loop (continues for total # of packets), receive any packet from the server. Then, save the received packet to its corresponding position in the window/packet\_buffer (Example, packet 1 received, store in position 1, even if packet 0 not received yet). Afterwards storage, the client then checks the packet position against the next expected packet. If the packet received is the same position in the window as the expected packet, then it increments the total count, increments the next\_packet\_received (within range of 0-5).

At this point, the client sends an ack of the next packet expected (Example, packet 1 received before packet 0, sends ack for packet expected at 0 rather than itself as writing it before packet 0 would corrupt the file)

**Server Side:**

When it finds a connection, pipes off a handler to deal with the specific client. Sends over basic file information (present, not present, size, etc). When the client returns a message indicating it is ready for the file information, the server will begin by reading the first 5 packets and their positions into a dict structure. When all 5 are read in, the server then sends the first 5 packets. As no ack is received yet, the next\_packet\_send value is still set to 0 rather than 4.

After sending the first 5 packets, the server waits for an ack (any ack) from the client. When one is received, it checks the value against the next expected packet (next\_packet send variable). If these are equivalent this implies the server has received the correct packet and the server increments the next\_packet\_send value to catch the next ack. The server then replaces the previous packet in the array with the next bit of the file to be read (Example, packet 0 ack received and packet 0/total replaced with packet 5/total in the window). The next\_packet\_send value is kept within 0-4 to remain within the window for the array

If the ack is not equivalent to next\_packet\_received, for example the ack for packet 0 is lost and 1 is received instead, the next\_packet\_send will not increment and still be awaiting the ack for packet 0 to ensure proper order.

**Struggles:**

**Alex:**

The python code was attempted as a last minute effort to create a working program. This was a struggle because we didn’t have much time to implement the file transfer with all the requirements. Python is also a new language for me so that was also a struggle just to understand what was going on.

**Josh:**

I agree with Alex in which the python code was a rushed fallback for the c code which was not working 100%. The reliability portion of the Python code was an obstacle.

**Tristan:**

Usage of pickle in Python over a UDP socket created errors. In order to get the packet position to the client easily, made use of a dict that created problems.

Couldnt find another way to modify the data structure in order to send across udp without issue from pickle.