**Team Big Endian’s Protocol**

**Introduction**

Our protocol is an application layer protocol that allows three network entities, a controller, a renderer, and a server, to communicate. With TCP connections, this protocol allows for the streaming of text files with basic commands like pause, resume, and playback between the entities.

**Purpose**

The purpose of this protocol is to provide a simple text file stream interaction between hosts. It’s not fully developed so it’s only practical as a learning experience.

**Overall Operations**

*Controller*

The controller allows for user input commands to make requests to the other hosts. The controller is the only host that allows any interaction between the user and the other network.

The controller is allowed to make a number of requests (as shown by command *!help*):

*Available commands:*

*!help : list available commands*

*ls-local : list local files*

*ls-remote : list remote files*

*play ‘filename’ : play remote file to Renderer*

*Functions: used for Renderer*

*resume : Resumes Renderer after paused*

*pause : Pauses Renderer*

*playback : Starts file from beginning*

*exit : Terminates program*

*Server*

The server is the backbone of the protocol. The server sets up all socket connections and handles a majority of the requests. The server is also the source of streaming file data.

Action after requests:

When client request *ls-remote*, the server will traverse its current directory and return the list of files available.

When client request *play ‘filename’*, the server will first make sure the file exists in its directory. After, it will begin streaming data line-by-line from to the Renderer.

When client request *pause* (only after *play* command is successful), the server will stop sending data to the Renderer.

When client request *resume* (only after *play* and *pause* command are successful), the server will continue sending data to the Renderer where it left off.

When client request *playback* (only after *play* command is successful), the server will restart the stream, resending data from the start of the file.

*Renderer*

The renderer is the simplest of the three hosts, only receiving data to be streamed from the server. The renderer begins by making a connection to the server and just listens to receive and print the data streamed.

Extra Notes:

* Run programs using *python3*
* To clarify ‘filename’ does not include the quotations, you just need to put filename name with the extension.
* Server only supports steaming of text files (extension .txt).
* Commands resume, pause, and playback only work after a file is streaming.
* Commands are case sensitive.
* Controller is the only host that takes user input request, hence the prompt symbol.
* ls-local prints the list in the proper format. ls-remote prints the list in an incorrect way. The issue happens when the controller receives the message, either from encoding or decoding.
* The program uses “10.0.0.1” as the server address when launched on mininet but should be changed to “localhost” when testing off mininet.

\*\*\*As you can see from our minimal amount of documentation for our protocol, we were more concerned about our implementation, how to create it, and how to actually run it on a network application. Not knowing how to do it pushed our priorities to it. We spent most of our time designing and coding it which led to very little documentation. In turn though, we were able to implement all the requirements for the network application and easily create a video walkthrough. That isn’t to say there couldn’t have been many more things to implement like handling disconnections, optimizing stream speeds, implementing actual messages with ACK returns, error detection/prevention/handling, etc, but time was an issue. We understand that the protocol documentation should’ve been our first priority. I hope this at least explains for the lack of documentation and maybe result in some leniency.