University of Texas at Dallas

Project Report

Joah George

CS 4390 Computer Networks

Dr. Nhut Nguyen

August 3, 2023

Table of Contents

[Program Description 3](#_Toc141734728)

# Program Description

Using Mininet, two virtual hosts are started with a switch connecting them. One host acts as a server and the other acts as a client. After the server is started, the client can be started. The client will connect to the server using TCP. Once a TCP connection is setup between the server and client, the client will show the operation list. In the operation list, there are three operations to select from – PULL, PUSH, and EXIT – and a brief description of each. More on each operation later.

## Device List

The device list displays pertinent information about the devices on the server and allows the user to select any number of them. The device list will be populated after the connection is set up but before the operation list is shown. After the TCP connection is set up, the client asks the server for the device information by sending a special PULL message. When the server receives the special PULL message, it sends information about all the devices it currently has. The client then creates a device list that lists the devices’ names, descriptions, and type. In the device list, a user can select any number of available devices by inputting each name with a space between them. If the user provides a name that matches no device’s name, it will be noted that the name was not found and continue execution. If the user provides no names or no names where found, then the client goes back to the operation list.

## PULL Operation

The PULL operation acts similarly to HTPP’s GET operation. When the user selects the PULL operation, the device list will appear (see more in [Device List](#_Device_List)). The client will send a PULL message to the server with the selected devices. The server will send a REPLY message containing the device status message for each device selected. In the client, each device will have its own line that displays the device’s name, type, description, and status.

## PUSH Operation

The PUSH operation acts similarly to HTTP’s PUT operation. When the user selects the PUSH operation, the device list will appear (see more in [Device List](#_Device_List)). The PUSH operation then prompts the user for the value he/she wishes to set each selected device to. Then, the client will send a PUSH message to the server where each device selected will have its status updated to the value input. After successfully setting the status of a device, the server sends a REPLY message to the client that contains the device status message. If the server fails in setting the status of a device, then an ERROR message will be sent to the client. The ERROR message will contain the device’s name, type, description and the value that it could not be set to. When the client receives a REPLYor an ERROR message, the value of the message will be displayed in blue or red, respectively.

## EXIT Operation

The EXIT operation will stop the program. When the user selects the EXIT operation, the client will send an EXIT message to the server. Then, the client will close its socket and terminate. When the server receives and EXIT message, it will close the client connection and its socket then terminate.

# Challenges

There were many challenges faced while making this project, but I will highlight the three that I believe were the toughest to overcome. The first major challenge was to establish the TCP connection between the client and server in Mininet. While I have done multiple projects previously that require a TCP connection, this was the first time I have used Python and Mininet. The primary difficulty was that I was not sure how to get the IP address of the hosts in Mininet using python. This was because when the gethostname function is used in Python, the hosts all shared the same hostname, and subsequently, they all shared the same IP address when the gethostbyname function is used on the shared hostname. For some reason, the hosts in Mininet were sharing a host name despite the ifconfig command showing different IP addresses. I eventually found an example of a Python TCP socket binding where the IP address field is left blank. I tried a blank IP address for the host and used the IP address in from the ifconfig in the client to connect to the host. This is the way that I was finally able to set up the TCP connection.

The second major challenge was designing a message that would fit the project requirements. The first message design was working wonderfully until I reread the project requirements and realized that there were multiple devices of a single device type (two light switches). While a redesign would have probably been the easiest solution, I didn’t want to completely scrap all the code I had already made with the first message design. I decided to modify the code to accommodate multiple devices with the same device type. This unnecessarily complicated the code, message creating, message parsing, and the server’s use of the message.

The third major challenge was some of the message were disappearing. When a user submitted a PULL or PUSH operation with multiple devices, the client would only get some of the messages. I checked the server, and the server was sending all the messages, but the client wasn’t receiving them. I noticed that when another PULL or PUSH operations was submitted, I would get the message of the previous operation. The messages weren’t completely disappearing but rather delayed. I can’t remember exactly how I found this out, but the cause was the server sending messages too fast to the client. I introduced a small amount of time between the servers messages and the messages were all displaying now.

## Learned

My primary takeaway from this project was that the TCP connection in socket programming is a powerful tool. This package allows programmers to very easily create top-level connections between servers and clients while also allowing anything to be send through the connection (once you figure out how to set it up properly). In Python, everything (objects, lists, dictionaries, etc.) can be easily converted to a JSON object and the JSON object can be easily converted to a string and sent through the TCP connection. Socket programming is a very powerful tool that reduces complexity of a program to enable faster and easier development of the functionality of the program rather than the underlying connections.

# Algorithms and Techniques

A technique I used to keep everything easy to understand was to always use a class for objects instead of a dictionary. This allowed my IDE to identify any syntax problems when I type the code instead of when I run the code. Classes also provide the ability to use inheritance. In the project, there are two devices – light switches and thermostats. With simple inheritance, I was able to store the fields that they shared in a single super class rather than on each of the subclasses. Additionally, the inheritance provided the base information needed to populate the device list.

A technique that I have used previously but did not use for this project was threading. Normally, when a server accepts a client, the server forks or threads so that it can continue to listen for more incoming clients. However, since there was only a single client that would be connecting, I decided to not use threading and keep the project simple.

# Suggestions

If the purpose of the project is to create the application layer protocol, then I believe that the students should focus on that. As mentioned before, I had trouble creating the TCP connection. Troubleshooting the TCP connection took up a lot of my time, but the purpose of the project was not to only set up the connection. I think that some skeleton code should be provided to the students so that they can focus on the application layer design rather than the TCP connection.