Jose Gomez
Kenny Ta

Assignment 1 Term Project Write Up

Group M: Morning May 5 2021

Jose:

For this lab the first thing we did was try to figure out how to get our IP. We decided to use Python since it is a language we have been using a lot lately. This took us a bit, but we managed to figure it out. The next step was to try to figure out how to get all the IP's in the local network. This was harder, mainly because i tried several things and it didn't work. Not only that but my laptop did not show up on the arp table in my Windows desktop. I downloaded Angry IP Scanner and my laptop still didn't show up. After some research I learned that the issue is because of how my network is set up. Changing this was not an option so I decided to try using a Virtual Machine.

This turned out to also be a frustrating endeavor. I had a Linux VM already set up for a different class. After messing with some settings, I managed to get my VM and the Host to see each other. There was however two problems with this solution. The first was that, for it to work, I had to set a static IP address. The second was that the program did not function as expected in Linux. I created a Windows virtual machine, however I was unable to get the host and the vm to see each other. This was very frustrating since I couldn't start coding until I had things set up correctly. (Later I found out this was not entirely necessary since the node could communicate with itself.)

I decided to do some more research into VM networks and found that virtual box has a thing called Nat Networks. This type of network simulates two VM's connected to the same router. This seemed like the perfect solution, so I cloned the Windows VM and set up a NAT network. I ran the arp table and both the nodes showed up on the others arp table. With this I was ready to test the little bit of code that we had. The Nat network ended up being very annoying since the VM's dont have internet access. I solved this by creating a shared folder where the files could be copied over.

At this point we had a simple script that would get the Ip, then it would take the first part of the ip and iterate through the last part, and check a specific port. This worked, but it was very slow, and we didn't have any sockets set up. After some research, I found out that we could use threads to significantly speed up the scanning process.

With only a couple weeks left, we decided to split up the work. I would work on the networking side of it, and Kenny would work on the file hashing, detecting changes in files, and anything related to files in general. We found some tutorials, which helped me get a better understanding of sockets. I created a server class and client functions. I also had to use threads to run the two servers since running more than one caused issues when they were running on the main thread.

Our node detection code worked as intended, so the next thing to do was to pass messages amongst different nodes. For this I used similar code to that of the tutorials, and I was

able to send the message “welcome to the server” back to the client. I created a while loop to test the function.

The next task was to send the file. Again I found a youtube tutorial to do so, and the final code was pretty similar with some differences. One of the main differences was the request code. The client socket closes everytime the connection ends, so changing the function of the socket was not a problem. The server socket, on the other hand, runs almost the entire time the program is running. In order for the server to know what It needed to do I created request codes. The client will send an int, and depending on that int, it will do something different. For example request code 0 lets the server know that the client wants to greet the server, so the server send back a greeting. Once this was done I was able to send the file across nodes.

The final task was to check if a node should be send a file. The way we dealt with this was relatively simple. When the program start is creates a dictionary with the file as the key and the hash as the value. The client sends the filename, the hash, and the modified time. If the file is not in the server’s dictionary, then the file needs to be added. If the file is in the dictionary, then check the hashes. If the hashes are different and the client’s modified time is more recent then the server’s modified time, then the file needs to be sent. The server will then send back its response, either true or false, letting the client know whether the file needs to be sent or not. While it doesn’t seem to complex, this code ended up giving me a lot of problems. After a lot of debugging and testing, I was able to get it to work consistently.

.

Kenny:

I dealt with the file hashing and everything else related to files so Jose can have an easier time implementing the service.

Upon trying to figure out how to get the hash in python, I saw that python has a built in hash function already. Therefore I did some tests to run the hash function in python and soon realized that it did not work. It kept hashing the exact same file differently. This at first did not make sense to me because it is the same exact file, why was it a different hash? Upon googling this I found that the hash function in python has a rng (random number generator). This random number generator kept creating the different hashes for even the same file. It was implemented as a security fix to bugs where hackers used the same algorithm to crack hashes / messages. That was interesting to learn about. So upon realizing I could not utilize Python’s hashing function, I kept googling different methods and then I found another library that I thought I could use (hash-lib). Upon reading the stackoverflow post, it stated that the hash would be the same for certain files unless I changed the file contents. Viola! I found the jackpot, but then I quickly realized it did not work because the way I was hashing it with MD-5 encoding, the files I tested were text files which worked but other files that I tested to hash did not such as the python.py file. Thus I googled some more and upon hours of searching, found another method that does hash all kinds of files in addition to storing them in the same hash if the files have not changed. This SHA-1 method we found works with both python files, text files and pictures.

Jose Gomez
Kenny Ta

Finally after getting the hash we needed different methods to run checks and make our assignment work. The first method we needed was a means to store the new hash so it can be easily accessible. I then thought about using a dictionary to store the hash and the key value would be the file so it coincides with the hash value. By doing so we would always have the hash allocated in the stack and easily reachable for calculations and comparisons. After creating the dictionary we needed a check to see if the hash matches any values in the dictionary. This is crucial for our assignment because when it checks for a matched value, it means that the files are already the same and therefore need to be updated.

The other methods that were made were a hash_files one that basically hashes all the files in the directory we are in and stores it into the dictionary. This is used at the beginning so whenever the program is run we would have the master copy of the files stored in the hashes. Upon changing the files, the program would check for different hashes and then perform the appropriate tasks if there is a change.

In addition to this, we implemented a check_changes method that there are changes we would grab the time modified and then append it and store the changes. This method took a couple of hours to implement along with the store_changes method because we could not get the time modified to be correct / I did not know how to grab the time modified. I at first wanted to experiment with the Timer library but that did not work and then upon googling, I found that we can get the getmtime for the file and that would be our time modified. Therefore after grabbing the information, we are able to differentiate what files were changed / modified.

Finally I implemented a looper function to check periodically for files that have changed. This was the most different function to implement and took many many hours to find and debug and create. We had to create a thread that loops recursively the same function over and over on a set interval, we set it to every 30 seconds it will run a file checker. After checking the files if it changed, we then proceed with the appropriate actions. This took a while because not many places have good answers on google on how to do it. A lot of places recommend we use a sleep() function but that does not work with our code. Therefore it took more research to find a solution.