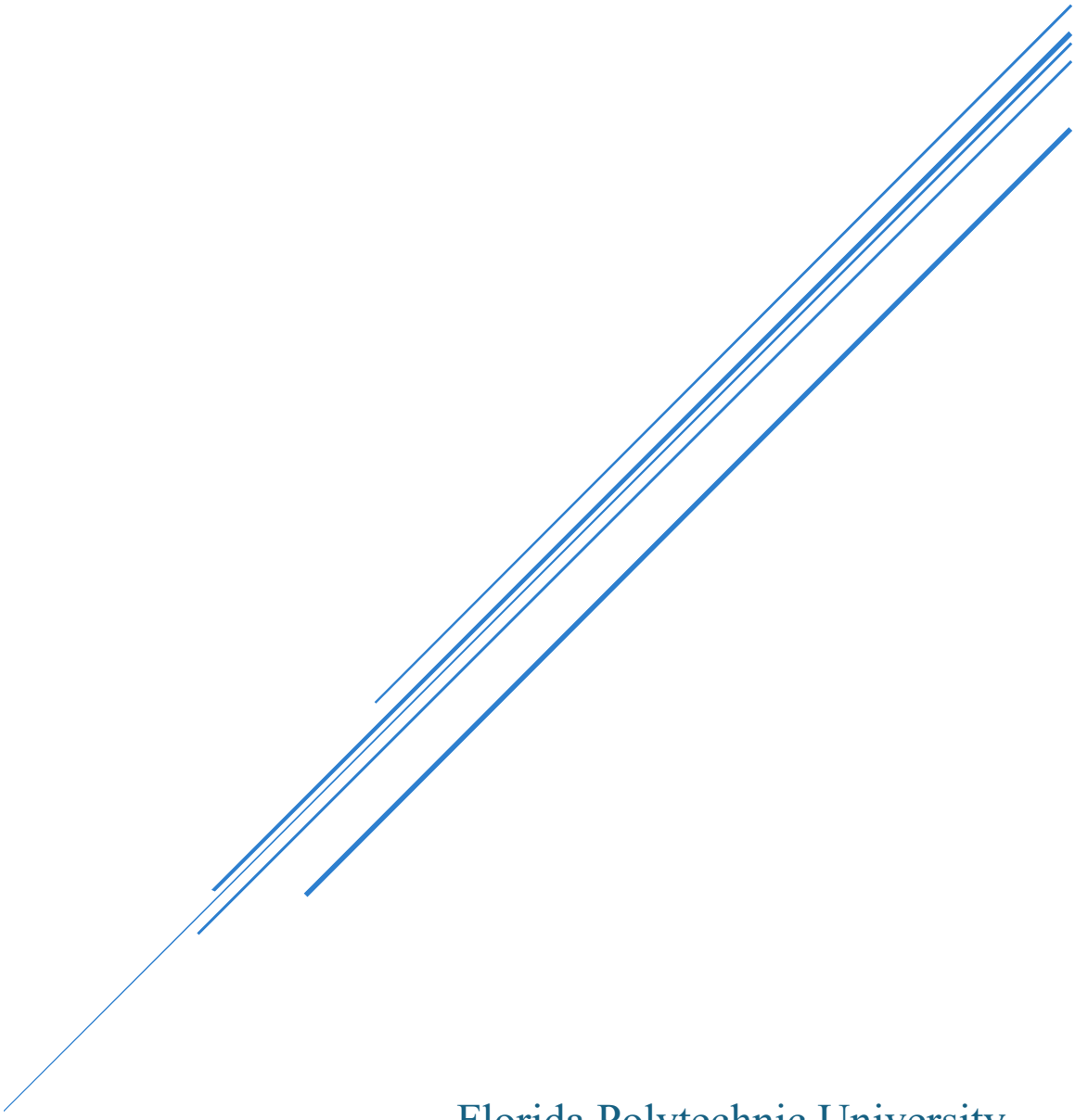
FINAL PROJECT

Natasha Linares, Dennis Carey, Joel Figueroa

GitHub: https://github.com/NatashaL2191/CNT3004FileSharing



Florida Polytechnic University
CNT3004

Introduction/Project Objectives

This Project is focused on the development of a simple client-server communication system using a Python Socket Library. The goal is to simulate how real-world networks handle authentication, data transmission, and connection management between clients and servers. The system is designed to: Establish a reliable TCP connection between a client and a server, allow a secure login authenticator system using hashed passwords to ensure encryption and privacy, enable message-based communication and commands, and collect and visualize network performance metrics such as latency and throughput. This project demonstrates practical networking principles like socket programming, data transfer, threading, and network monitoring.

The main initiatives of the project include:

1. *Authentication System Implementation*
   -Developing a login mechanism that verifies clients before allowing the connection between client and server.

2. *Client-Server Communication*
   -Enable a two-way communication of the TCP connection where clients can send commands and receive responses from the server.

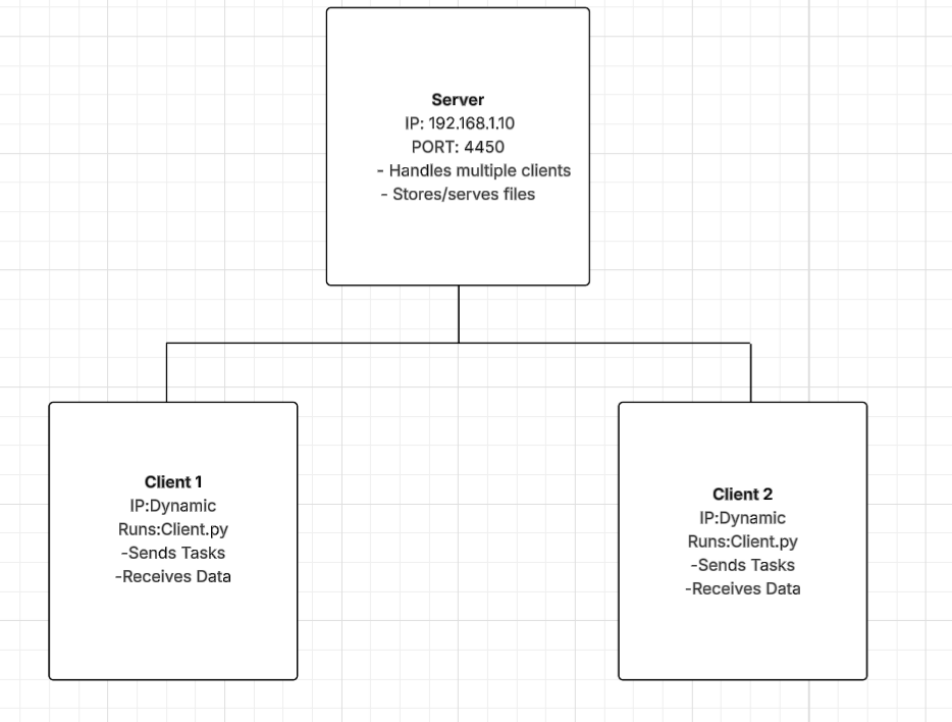
3. *Concurrent Client Handling*
   -Use threading to allow the server to handle multiple client connections simultaneously.

4. *Performance Measurement and Visualization*
   -Implement a separate testing module that measures the overall performance metrics of the connection and graph it continuously.

5. *Multi-Device Support*
   -Design the architecture so that the system can run across multiple computers on the same network.

6. *Secure disconnection and Error Handling*
   -Ensure proper disconnection of client connections, handle authentication failures, and prevent socket crashes.

System Architecture



This diagram illustrates the client-server network architecture used in the system. The **server**, operating on IP address 192.168.1.10 and port 4450, acts as the central node responsible for handling multiple client connections simultaneously. It manages data storage and serves files or responses upon client requests. **Client 1** and **Client 2** both have dynamic IP addresses and run the Client.py script to connect to the server. Each client sends tasks or requests to the server and receives corresponding data or responses in return. This setup demonstrates a typical multi-client server model where the server coordinates communication and data exchange between multiple clients efficiently.

File Tree Architecture

Server/:

Within the server file, the main server implementation python file is stored. The "server_data" folder is where uploads from the clients are stored within the server. The authentication file contains the function that handles the login process utilizing the "users.json" file to check users allowed to connect to the server.

Client/:

Within the client file, the main client implementation python file is stored. This file contains the main functions of the client including downloading, uploading, login and deleting files. The folder acts as the main area where the clients communicate with the server.

Tests/:

The Tests folder acts as the analysis folder, containing the graphing function file "test_analysis" that in real-time measures the latency of the packets sent from each client to the server. The server and client files contained in the Tests folder act as practice runs to ensure the program is running correctly and to ensure changes could be precisely tracked and reverted if necessary.

Implementation Details

The implementation of this client-server system is divided into three main parts: the server, the client, and the performance measurement module. Each component plays a key role in the performance of the system.

Server Side

The server module includes the main server Python file, which implements the TCP connection to establish communication between the server and multiple clients. The server defines key connection parameters such as the IP address, Port number, Address format, Buffer size, and Data format to ensure proper message handling between both ends. This setup allows the server to listen for incoming client requests and manage multiple sessions efficiently. The server file also includes the logic that determines how to handle specific commands requested by the clients, such as **Login**, **Logout**, and **Upload**, as well as sending acknowledgements for each request to confirm successful communication and actions.

Within the server module, the **authenticator** plays a critical role in managing user access and security. The authenticator is a separate file stored within the server directory that defines the function responsible for verifying user credentials. It compares the unhashed passwords entered by clients with the stored, hashed passwords to ensure security and prevent unauthorized access. The authenticator references a “users.json file”, which contains a structured list of valid usernames and their corresponding hashed passwords. When a user attempts to log in, the authenticator checks this file to validate the credentials before granting permission to connect.

This authenticator is then seamlessly integrated into the main server file, allowing it to operate in real time alongside other server processes. This integration ensures that user authentication happens immediately during the connection phase, resulting in a quick, secure, and efficient login experience. By separating the authentication logic into its own file, the system maintains a modular design, making it easier to update security protocols, manage users, or extend functionality without disrupting the core networking code of the server. Overall, this design structure provides both reliability and scalability, ensuring that the server can securely manage multiple client connections simultaneously.

Client Side

The client module includes the main client Python file which establishes the TCP connection to communicate with the server. The client defines important connection variables such as the server IP address, Port, and Data format to ensure compatibility with the server's configuration. This module handles all user-side interactions, including sending commands to the server and processing responses received. The client allows users to perform actions such as **Login**, **Logout**, and **Upload**, and waits for acknowledgements or data sent back by the server to confirm successful execution of each request.

Within the client module, several helper functions are defined to handle specific tasks and improve user interaction. These include functions for encoding and decoding messages, sending structured data packets, and managing file uploads. The client interface is designed to send properly formatted requests that the server can easily interpret and respond to. It also includes basic error handling to notify users if an action fails or if the server connection is interrupted.

This design ensures smooth and consistent communication between the client and the server, maintaining a reliable connection throughout the session. By separating user actions into defined functions, the client module remains well-organized and easy to expand upon, allowing new features or commands to be added in the future without disrupting existing functionality. The client's lightweight design also ensures that commands are executed efficiently, providing users with a quick and responsive experience when interacting with the server.

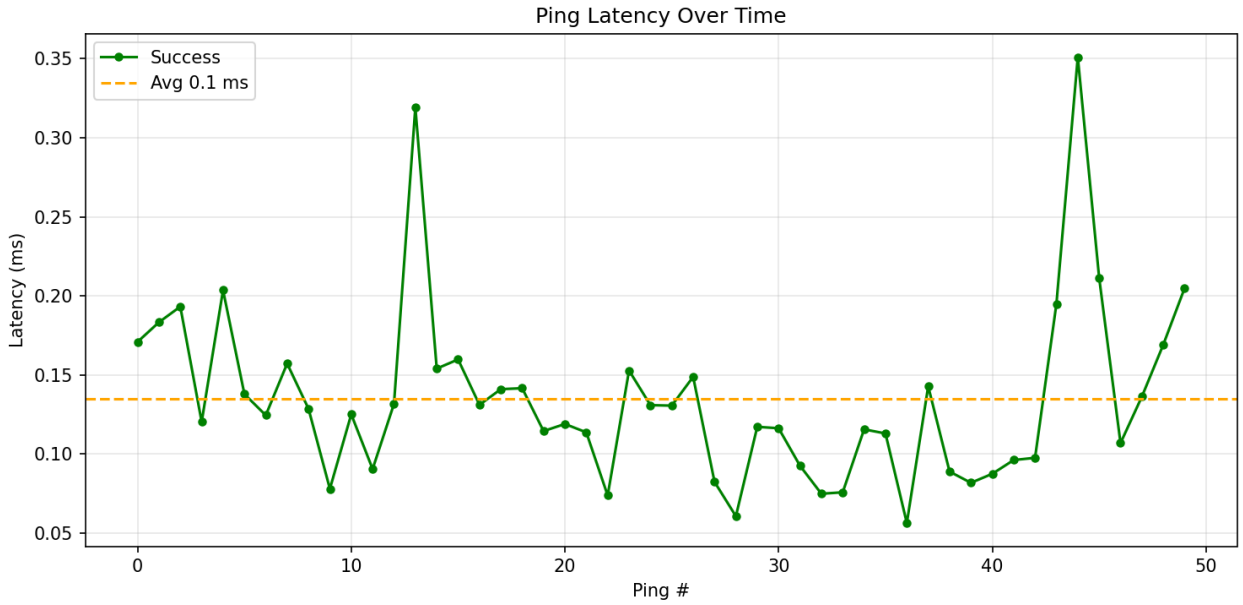
Performance Module

The performance module focuses on analyzing and evaluating the system's efficiency during operation. This module is responsible for monitoring different performance metrics such as data transfer speed, response time, connection stability, and overall resource usage. It collects and measures these values during client-server interactions to determine how well the system performs under various conditions.

The performance module includes functions that record the start and end time of each operation, calculate the latency between client requests and server responses, and measure the average upload and download speeds. These values are then displayed and logged to help identify potential bottlenecks or inefficiencies in the communication process. By continuously tracking performance data, this module helps ensure that the network maintains consistent and reliable functionality even under heavy use.

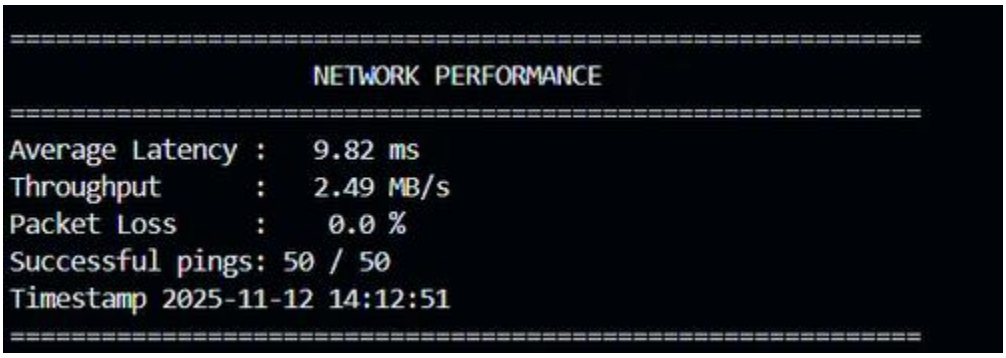
This module is integrated with both the client and server components, allowing it to operate seamlessly in the background while the system runs. The collected data can be used to make informed improvements to the system's code, optimize resource management, and enhance the user experience. Maintaining a dedicated performance module also helps preserve modularity in the project's structure, keeping testing, optimization, and analysis separate from the main communication logic while still being easily accessible for future development or debugging.

Experimental Results and Analysis



The provided graph measures the latency of each ping throughout the connection between the server and the client. The peaks in the data reflect uploads and downloads of larger file types. Although, through the graph it is indicated that the uploads took a bit more time than average.

This behavior can be explained by the nature of network data transfer and how bandwidth is allocated. Uploading generally involves sending data packets from the client to the server, which can be slower due to limitations in the client's upstream bandwidth or higher processing overhead. Each peak represents a moment when the network was handling a heavier load, causing temporary increases in latency. These fluctuations are normal in real-world network environments, especially when multiple data transfers or larger file operations occur simultaneously. Overall, the graph demonstrates how network congestion and file size can directly affect response time and the efficiency of data transmission between connected devices.



The network performance summary provides an overview of how efficiently data is transmitted between the client and server. The **average latency** of 9.82 milliseconds represents the average round-trip time for packets to travel from the client to the server and back, calculated by dividing the total ping time by the number of pings sent. The **throughput**, measured at 2.49 MB/s, indicates the rate of successful data transfer over the connection and is found by dividing the total amount of data transferred by the total transfer time. The **packet loss** of 0.0% shows that no data packets were lost during transmission, which is calculated using the formula $(\text{Packets Sent} - \text{Packets Received})/\text{Packets Sent} \times 100$. The **successful pings** value of 50 out of 50 demonstrates a fully stable connection with all ping requests successfully receiving replies, confirming the reliability of the network. Finally, the **timestamp** (2025-11-12 14:12:51) indicates when the test results were recorded, providing a time reference for the performance data. Together, these metrics show a fast, consistent, and lossless connection between the client and server.

Average_Latency_ms	Throughput_MBps	Packet_Loss_percent	Successful_Pings	Total_Pings	Timestamp
inf	0	100	0	50	2:04:55 PM
Average_Latency_ms	Throughput_MBps	Packet_Loss_percent	Successful_Pings	Total_Pings	Timestamp
0.117454	7.686396971	0	50	50	2:05:37 PM
Average_Latency_ms	Throughput_MBps	Packet_Loss_percent	Successful_Pings	Total_Pings	Timestamp
9.094025999	2.49539729	0	50	50	2:12:07 PM
Average_Latency_ms	Throughput_MBps	Packet_Loss_percent	Successful_Pings	Total_Pings	Timestamp
9.822475997	2.487793017	0	50	50	2:12:51 PM

This table records multiple instances of network performance measurements taken over time between a client and a server. The data shows variations in latency, throughput, and packet loss across different timestamps. The first test, recorded at **2:04:55 PM**, shows an infinite latency (inf) and 100% packet loss, indicating a failed connection attempt where no pings were successfully transmitted. The following test at **2:05:37 PM** shows a drastic improvement, with an extremely low latency of **0.117 ms** and a high throughput of **7.68 MB/s**, reflecting a stable and high-speed connection. The next two tests, at **2:12:07 PM** and **2:12:51 PM**, show consistent results with average latencies around **9 ms** and throughput around **2.49 MB/s**, both maintaining 0% packet loss and full ping success (50/50). These results demonstrate the network stabilizing after an initial failed attempt, maintaining strong reliability and performance in subsequent tests.

Problems Faced

Some Problems we have faced include:

Creating the authorization file

The authorization python file is used alongside the server python file that refers to a Json file containing hashed passwords and usernames. To get this working properly we had to make sure the passwords were exact when inputting them into the Json and server file along with the type of hash it was (md5). I had to make sure to run the server python file through the terminal rather than running through VsCode's "run" button.

Uploading Larger Files

We had trouble uploading files bigger than 80MB, when we attempted it would end up timing out. To solve this problem, we had to mess with the downloading function, splitting the data into chunks to make sure all the data wasn't uploaded to the RAM at once.

What was learned

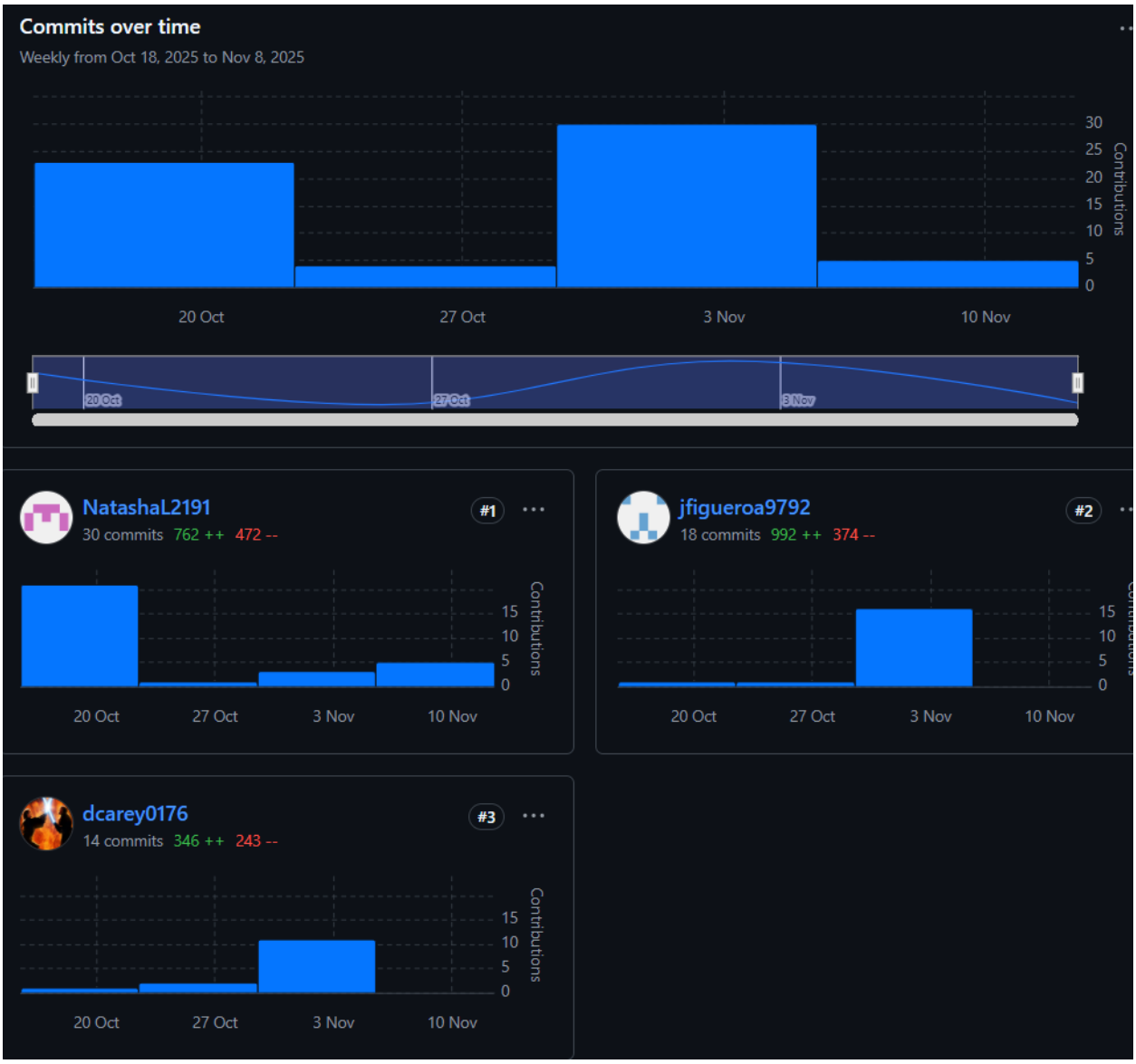
Natasha: Through this project, I have learned the fundamentals of socket programming, implementing IP, Ports, and what is necessary for a TCP connection to effectively work. Creating the authentication was a bit challenging but reflected the inner works of the program. Identifying the server's role in acknowledging users and passwords and the process of decryption to allow users to connect to the server. This project has also echoed the format of a group project. My group worked continuously and were dedicated to completing the project. I strive to work with more groups like this, bouncing off ideas and implementing a working script for a common goal.

Joel: Working on the client side of this project has been a really good learning experience. Before this, I didn't know much about how clients and servers actually communicate, but now I have a much clearer picture of how data is sent and received through sockets. Setting up the connection and testing commands like login, upload, and delete helped me understand how the client interacts with the server in real time. It was also cool to see how encoding and decoding messages is necessary for both sides to "speak the same language." Even though I ran into a few bugs and delays at times, troubleshooting those problems helped me see how small details in network communication can make a big difference. Programming the client also gave me a better sense of how to organize my code into smaller, reusable parts. I modified separate functions for each main command, like uploading, downloading, and deleting files, which made everything easier to manage and update. I also learned the importance of handling errors and checking for things like missing filenames or failed connections before moving

forward. Overall, working on the client, even though it wasn't from scratch, taught me not just how socket communication works, but also how to think through problems, test solutions, and be patient while debugging. It’s been challenging, but also really rewarding to see it all come together.

Dennis: As we worked on this project, I've learned that communication is key, not only for TCP connections but also for human interactions. The server and client constantly needed to send and receive messages to know what was going on. If the client connects to the server and never gets a response, then no further communication will happen. This is because the TCP connection will time out if the server or client is waiting too long before the next message is received. Before a TCP connection can be made, the server must be listening to a specific port. The combination of IP and the port is a socket. TCP needs two sockets to make a connection. Establishing the connection was the easier part of the process. Learning how to send large files was the trickiest part. Once we verified that small files worked, we had to test different host networks to see if the packets were being sent or if the internet speed was just slow. Sending the files in chunks was a good way to make sure the connection wasn't timed out, and the packets were still being sent. Using GitHub for this project made it possible for all of us to make changes to the files without having to manually send over different versions of the code to everyone. Each time the files were updated, there was a message that went along with it to know what changes were made, just like whenever the server and client communicated.

Individual Contribution Table



Conclusions and Future Work

In conclusion, this project was a thorough and effective representation of how socket programming is performed and utilized. The use of threading allows for a reflection of how computer networks perform in relation to multiple clients communicating with one server. The functions that were implemented actively test the system and its capability to accurately deliver the requests from each client or the server.

The authentication process was a significant process for the network. The use of the usernames and passwords as they were listed and encrypted then referred to by the server, and the process of authentication along with the login function, demonstrated the real-world capabilities of our network.

Communication through the client and server is essential to understanding how data is transmitted, received, and validated in a controlled network environment. By developing modules for testing latency, throughput, and packet loss, as well as commands for creating and deleting directories, this project simulated key elements of an operational network system. The logging and analysis of network performance data provided valuable insights into response times, data transfer rates, and reliability.

Overall, this project successfully showcased the fundamentals of computer networking, from authentication and message handling to real-time performance measurement. It not only deepened understanding of socket programming and client-server architecture but also demonstrated how these technologies are applied in real-world systems that depend on secure, efficient, and concurrent network communication.