ECE-369 Computer Networks
Mid-Term Progress Report

Michael Bisbano (Lead)
Michael Berek
Jacob Merola

0 GITHUB REPOSITORY

This project is hosted on Github. Project progress can be viewed at any time using the following link: https://github.com/mbisbano1/369_Project_1.

1 UNDERSTAND SOCKET PROGRAMMING

Experiment 1:

TCPClient.py and TCPServer.py were loosely based on the sample code, but largely rewritten to make use of Python object-oriented constructs like classes. The development of this portion of the project was rather uneventful, as a TCP client and server were already completed in Homework 3.

Experiment 2:

TCPServerMultithread.py was a bit more difficult to implement than the previously-completed TCPClient.py due to the way in which threading is implemented in Python. The server would start and accept connections as normal, but could not be shut down gracefully as the threads would continue to run. However, an email from the Professor explained that the server would only be started and never stopped, preventing this from becoming an issue.

Experiment 3:

The UDPclientTimer.py code was written based off of the hint given, using the UDP code developed for Experiment 1 as a starting point. Code for it can be found in the group GitHub repository. This was implemented without issues.

Experiment 4:

Peer2Peer communication apps have been taking longer to develop than expected. The UDPpeer.py file is complete and working, but only on Windows devices. The idea to implement a broadcast “ping” when the peer.py is run to search for other peers on the network allowed the program to work without needing the user to input a host IP address. The TCP counterpart for this is still in development, but both files can be found in the experiment4 branch of the group GitHub repository. When the TCPpeer.py is finalized, the experiment4 branch will be merged into the master branch.

Experiment 5:

UDPclientTimer.py was successfully ported to C. The code is much more verbose than its Python counterpart, as <sys.socket.h> requires three special structs to be declared. Input sanitization was added, as C does not handle bad input as gracefully as Python does with its try/except constructs.

2 MAKING DESIGN CHOICE: CLIENT/SERVER OR PEER-TO-PEER

The best choice for this internet application would be to implement a Client/Server architecture. Since the main goal is providing access to files after a user's credentials are validated, there will likely have to be a centralized server handling access rights and storing files anyways, so implementing P2P would likely just clutter up the flow of our network application and create more vulnerabilities.

3 INTEGRATING SOCKET CODES USING C

Experiment IV required the group to create a TCP socket program in a language other than Python. C was chosen because it is something we were already familiar with due to prerequisite classes like ECE-160 and 161. As of writing, this portion of the project is complete.

4 INVENTING OWN APP

Problem Statement Revision

No changes have been made to the problem statement. Our initial proposal was accepted by the professor as written, and we do not anticipate any difficulties with our proposal, so it has not been modified.

State of the Art: One Key Paper and Annotation

This project requires users to authenticate using their UMassD credentials. Accepting the password and moving it through the backend in plain text is unacceptable, as it presents a significant security risk. To prevent this, the password will be salted and hashed. The key paper for this project will focus on password salting and hashing. The annotated bibliography entry for this paper is shown below.

Bernstein, Daniel. "Cryptography in NaCl," March 2009.

<http://cr.yp.to/highspeed/naclcrypto-20090310.pdf>
This paper provides useful information about NaCl, a library providing cryptography services for C and Python programming languages. Although this library will likely not be used for this project, the paper still provides information about implementing cryptography elements such as public/private keys. This information is useful on its own and not tightly coupled with the library. This paper also provides security proofs of the implementation by demonstrating how the library is resilient to different types of attacks.

Technical Approach, Refined

Our technical approach has not changed significantly since the project was proposed. A system top-level diagram is shown in Figure 1. Please note that IP addresses and port numbers are subject to change.

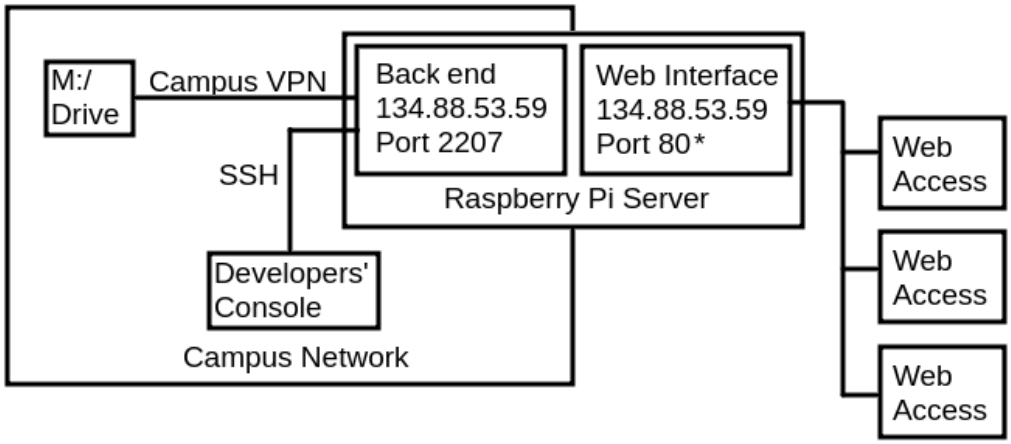

Figure 1: Top-level diagram

Preliminary Results Demonstration

We have accessed the M-Drive through the Raspberry Pi server and successfully transferred files end-to-end using existing internet applications (SSH and SCP) to validate the network path and ensure the end goal of our project is something that is possible. This has shown us that by using the Raspberry Pi server as an access gateway, we can get around the UMass Dartmouth VPN requirements, and allow any client anywhere access to the M-Drive files. One thing we may need to reconsider for the scope of the project is the possible effects this “unconventional” access into the M-Drive may have for our accounts. After using the Raspberry Pi server to access the M-Drive files for a bit, we experienced one of our student accounts being shut down for a brief period of time, which likely is an artifact of the CITS security disliking our behavior. To avoid this happening again, we propose to “simulate” the M-Drive as a folder full of demo information

on the host Raspberry Pi server. This will allow us to still develop our internet application and test its functionality without risking our access to all campus online resources being restricted, especially during a semester with many online tests and assignments due all the time.

5 INDIVIDUAL ACCOUNTABILITY

The team member roles remain unchanged from the proposal. The list of team members and their respective roles are listed in Table 1.

Table 1: Team member roles

Team Member	**Role(s)**
Michael Bisbano (Lead)	System integration, file/directory (inode) organization (Python, C)
Michael Berek	Front end development (HTML)
Jacob Merola	Back end development (Python)

6 NEXT STEP

The next step we plan on doing as a group is creating some core code as far as back-end and front-end development. After completing the experiments that we were assigned to look into and solve, we have a basic understanding of socket programming. As a result, we will have a much easier time implementing the methods used to solve these experiments into our application that we're creating. The only difference will be a couple things that will require some side research. These are things such as salting and hashing to add security to log-ins, as well as actual HTML expertise to create an easy readable, and most importantly usable interface for users. To go more in depth for each of us:

- **Michael Berek:** As the front-end developer of this project, my next step will be looking into starting development of the actual user interface. To do so it'll require more practice with HTML to make sure users can easily access the server.
- **Michael Bisbano:** As the system integrator and group leader, I will need to maintain communication throughout the group so that our developments can meet in the middle and work together seamlessly. I will also be implementing the file management and storage programming, which will likely lie somewhere between the two other portions of the code. It will be important moving forward to develop our own "protocols" so that our different pieces of code can communicate, so this is likely something I will tackle first to

set the framework for future developments.

- **Jake Merola:** My next step will be to implement the password salting/hashing component, as this will be the most difficult of my responsibilities. Once I can solicit credentials from the user and store them securely, I will then work on the interface between the Python client and the M:/ drive, which uses the campus Raspberry Pi array (RPi) as a gateway. This should not be too difficult as there are a myriad of different ways to interface with the RPi, such as SSH, SFTP, SSHFS, and FTP.