IEEE Project Report Template

Title Page

- **Project Title:** FTP Server
- **Authors:** Everette Webber, Michael Martinez, Gregory Dorfman, Sepehr Nourbaksh, Matt Sanchez
- **Affiliation:** California State University Fullerton
- **Course:** Computer Networks
- **Programming Language:** Python
- **Date:** 11/27/2024

Abstract

We build a File Transfer Protocol server. That is an application layer protocol capable of transferring files between two computers. We also built the server-side and client-side applications to handle this protocol. The result is that we can send files between any two computers running this on a Local Area Network. We utilized the Python programming language, specifically the socket library to help us along.

Keywords

Python, Computer Networks, Socket Programming, Protocol Simulation,

---

1. Introduction

1.1 **Background**

- The internet is nothing but computers talking to each other over a network. Therefore, it is crucial to understand how these computers talk to one another. One of the best ways of doing that is by building our own network application. That is what this project will do.

1.2 **Objective**

- The goals of this project is to develop a client and server application that can transfer files between one another using the windows terminal. The files will be sent over the Local Area Network using python.

1.3 **Scope**

- This project will cover designing the application layer protocol, building the client-side code, and building the server-side code.

---

2. Related Work

- Any device connected to the internet already has built in capabilities of handling the Transport, Network, and Link layers of the Internet. So, all we have to build is the application layer.
- We will be using the Python socket library.

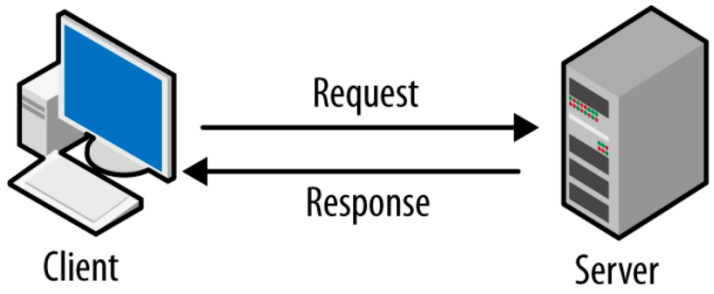
---

3. System Design and Architecture

3.1 **System Overview**

- At the highest level our system will have a server and a client. It will form two channels between these. One channel will be the control channel and be used to send commands back and forth. The other channel will be the data channel and be used to pass files back and forth.

3.2 **Architecture Diagram**



- Client
  - The machine that sends or requests files.
- Server
  - The machine that holds files that can be sent to.

3.3 **Modules**

- Client-side application
  - First attempts to connect to the server
  - Next waits for user input

    - Based on user input it sends a command to the server and waits for a response
        - Based on the command it could wait to receive a file, send a file, or wait to hear a file was deleted.
- Server-side application
    - First it starts listening for any clients
    - Next once it is connected to by a client it listens for any commands
    - Based on the commands it hears it response
        - This response could contain a file, waiting to receive a file, or a status about some change on the server.
- Communication protocol
    - The communication protocol uses the following datagram

Packet Number (2 bytes)	Command Name (6 bytes)	DataSize (4 bytes)
Data		

    - Where the Command Name tells the receiver what to do with the packet.
    - Based on the Command Name the Data may or may not be filled
        - Its most common use case is for files that are sent back and forth.

---

4. Implementation

4.1 **Technologies Used**

- **Socket Programming:** Python `socket` library

4.2 **Key Features**

- Establishing a TCP connection using the socket library
- Simulating a protocol like FTP
- Real-time data transmission between client and server

4.3 **Code Overview**

- Provide essential code snippets with explanations (e.g., socket creation, data handling).

4.4 **Algorithms**

- Because the Python socket library uses computers built in network capabilities all network layers other than the application layer was handled out of the box. Transport, Internet, and Link layers were already handled. This means we did not need to worry about routing, packets being dropped, or even fragmentation.

- The main algorithms we had to worry about where Finite State Machines to create the procedures for the client and server to follow when the received certain packets.

---

5. Results and Discussion

5.1 **Testing Scenarios**

- Send/Receiving data
    - Works as expected.
    - Multiple file types were tested out and were received by both the client and server as expected with no observable data loss.
- Handling errors
    - Both the client and server handle all known errors as expected
    - The biggest known error is when the client requests a file that does not exist on the server
        - This error is handled
- Multiple Clients
    - This was not a given requirement at the start of the project and thus it is not implemented.

5.2 **Performance Metrics**

- Latency: Minimal, appears to depend mainly on file size, which is to be expected
- Throughput: Can only handle one client at a time but appears to have no throughput issues otherwise

5.3 **Challenges and Resolutions**

- The two largest challenges in development were
    - Designing the state machines for the procedures
        - Once the server receives a packet requesting a file, how does it know the next series of packets it should expect?

        - This problem resulted in us having to make a general finite state machine to track what procedure any given server or client is in, what step they are on, and what packets they should expect next.
    - Storing data sockets as variables
        - Part of developing the state machines for the procedures ment saving what packet was expected on what socket for a given step.
        - As part of these we saved the sockets themselves as variables in the individual steps of these procedures, we thought these were references to global variables
        - Due to Python being a dynamically typed language and all signs pointing to the fact that the global variables where being saved in the procedures, we still had to redefine all the procedures any time a socket was changed.

5.4 **Discussion**

- Strengths
    - It works
- Weaknesses
    - At the moment the client is capable of telling the server to uninstall the server.

---

6. Conclusion and Future Work

6.1 **Conclusion**

- We designed our own FTP application layer protocol. We also built working client and server code to use this protocol.

6.2 **Future Work**

- Main areas of improvement
    - To get it working over more than just a LAN
    - To make it so the Server can’t uninstall itself
    - To make it so the Server can handle multiple clients at once

---

7. References

- See the Application Layer Protocol.pdf for more info on the protocol
- See the README.md for how to run the code

---

8. Appendices

- N/A

Example Sections for Python Networking:

Key Implementation Example:
python
Copy code

```
# Simple Python socket server example
import socket

# Create a socket object
server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

# Bind to a local address and port
server_socket.bind(('127.0.0.1', 8080))

# Start listening for connections
server_socket.listen(5)
print("Server is listening...")

while True:
    client_socket, addr = server_socket.accept()
    print(f"Connection from {addr}")
    data = client_socket.recv(1024).decode()
    print(f"Received: {data}")
    client_socket.send("Acknowledged".encode())
    client_socket.close()
```