Report: TCP File Transfer Application

Student Name: [Nguyen The Thinh]
Student ID: [22BA13295]

1 Introduction

The objective of this project is to design and implement a peer-to-peer file transfer application using the TCP/IP protocol suite. The system consists of a Server (sender) and a Client (receiver), implemented in C++ using the Linux Sockets API. The primary goal is to ensure reliable data transmission over a network connection.

2 Protocol Design

To avoid ambiguity in data transmission, the protocol follows a strict sequence of events. Instead of a simple request-response model, the interaction is stateful.

The communication flow is illustrated in the Sequence Diagram below (Figure 1). Time flows from top to bottom:

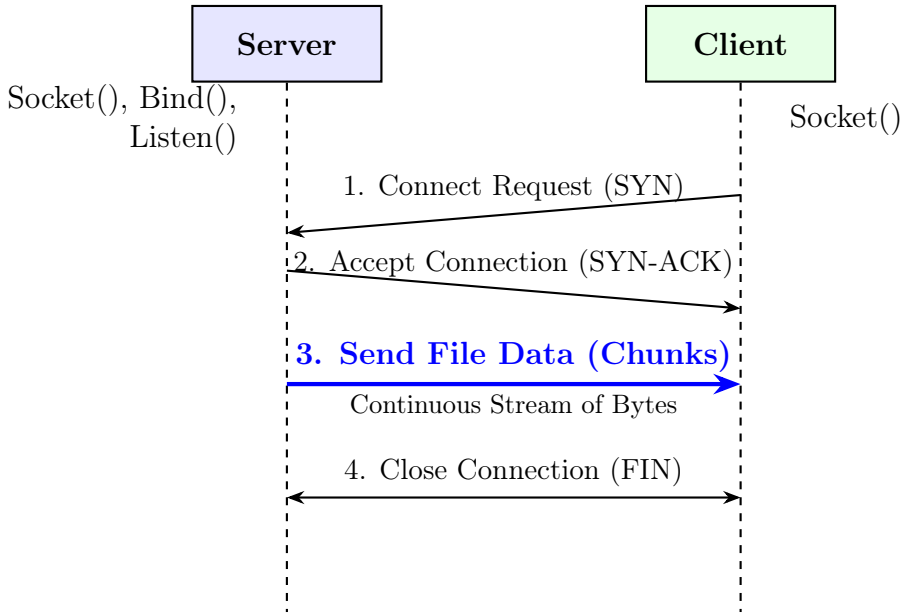

Figure 1: Protocol Sequence Diagram

3 System Organization

The system architecture follows a Client-Server model. The two components interact through the Operating System's transport layer.

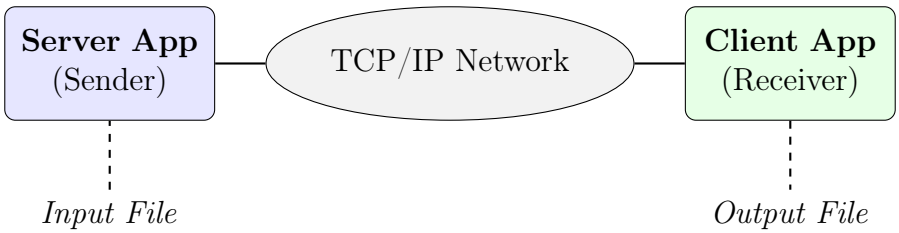

Figure 2: System Organization and Data Flow

4 Implementation Details

The implementation relies on blocking sockets. Below are the critical snippets handling the file transfer logic.

4.1 Server Side (Sender)

The server reads the file in binary mode and sends it in chunks of 1024 bytes. We use `gcount()` to ensure only valid bytes are sent.

```
// Open the file
ifstream fileToSend(filename.c_str(), ios::binary);
char buffer[BUFFER_SIZE];

while (fileToSend) {
    fileToSend.read(buffer, BUFFER_SIZE);
    streamsize bytesRead = fileToSend.gcount();

    // Only send the exact amount of bytes read
    if (bytesRead > 0) {
        send(clientSocket, buffer, bytesRead, 0);
    }
}
fileToSend.close();
```

Listing 1: Server Code Snippet

4.2 Client Side (Receiver)

The client connects to the server and enters a loop to receive data until the server closes the connection (returns 0).

```
// Create output file
ofstream receivedFile(saveFileName.c_str(), ios::binary);
char buffer[BUFFER_SIZE];
int bytesReceived;

// Receive loop
while ((bytesReceived = recv(clientSocket, buffer, BUFFER_SIZE, 0)) > 0) {
    receivedFile.write(buffer, bytesReceived);
}

receivedFile.close();
```

Listing 2: Client Code Snippet

5 Conclusion

The TCP file transfer application has been successfully designed and implemented. The use of TCP sockets guaranties data integrity during transfer. The sequence diagram clearly defines the interaction states, and the modular code structure allows for easy maintenance and future upgrades (e.g., adding multithreading).