Practical Work 1: TCP File Transfer

Nguyen Minh Tuan
23BI14441

November 21, 2025

1 Introduction

The goal of this practical work is to implement a one-to-one file transfer system over TCP/IP using the Command Line Interface (CLI). The system consists of a single server and a single client interacting via Java Sockets.

2 Protocol Design

The protocol ensures the server knows exactly what it is receiving before the content arrives. The flow is:

1. **Connection:** Client connects to Server (Port 5000).
2. **Metadata:** Client sends File Name (UTF) and File Size (Long).
3. **Data:** Client streams bytes; Server reads until size is met.
4. **Feedback:** Server sends a success message.

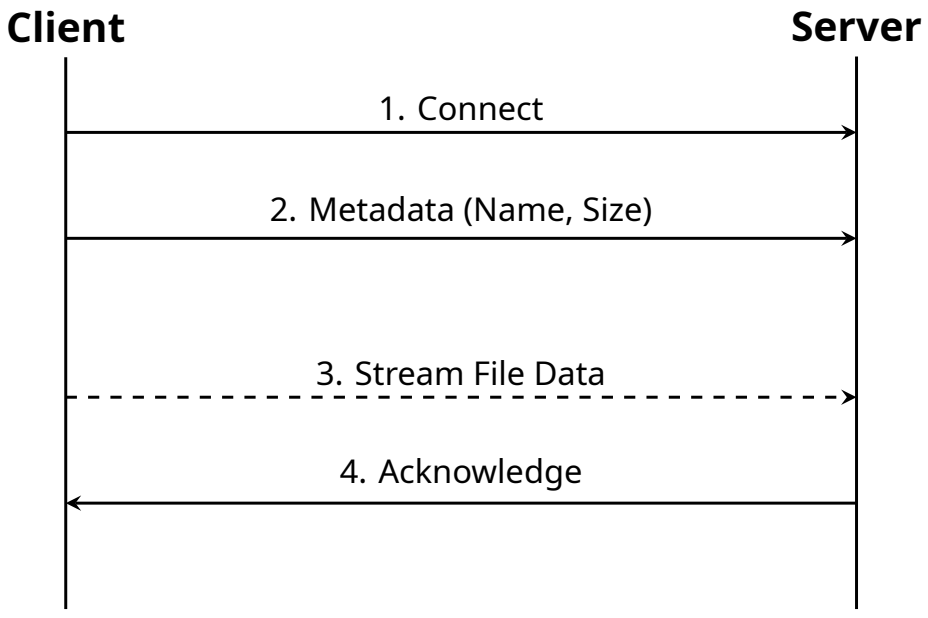

Figure 1: Protocol Design Sequence

3 System Organization

The system follows a Client-Server architecture using Java Sockets.

- **Server:** Binds to port, listens, and accepts connections.
- **Client:** Creates socket, connects, and writes data.

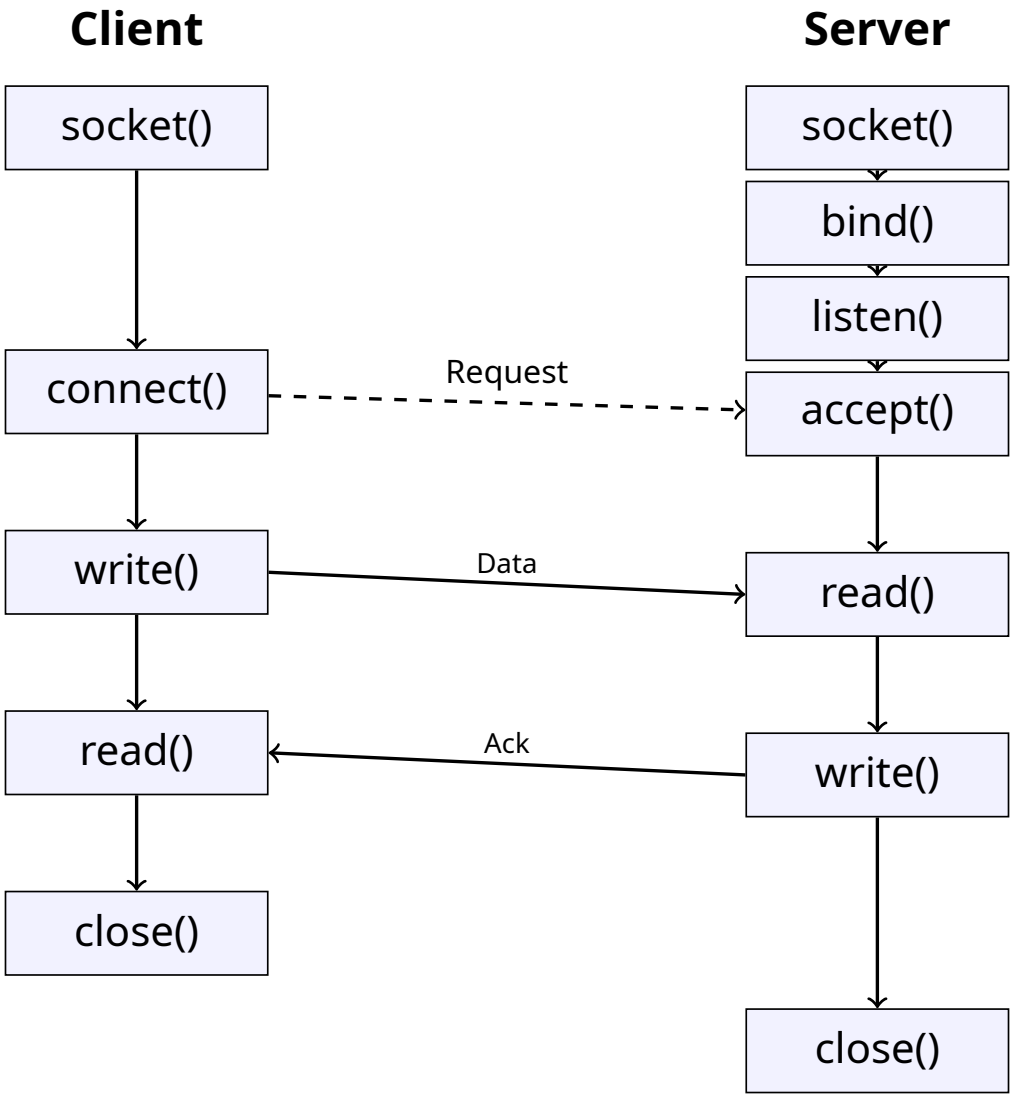

Figure 2: System Organization (Socket Lifecycle)

4 Implementation

The implementation uses `java.net` for networking and `java.io` for file streaming.

4.1 Server Implementation

The server waits for a client, reads the file name and size, then streams the content to a local file.

```java
package practice_1;
import java.io.*;
import java.net.*;

public class Server {
    public static void main(String[] args) {
        try {
            ServerSocket ss = new ServerSocket(5000);
            System.out.println("Server Waiting...");
            Socket s = ss.accept();

            DataInputStream dis = new DataInputStream(s.getInputStream());
            DataOutputStream dos = new DataOutputStream(s.getOutputStream());

            // 1. Read Metadata
            String fileName = dis.readUTF();
            long fileSize = dis.readLong();
            System.out.println("Receiving: " + fileName + " (" + fileSize + ")");

            // 2. Prepare Destination
            File fileDest = new File("received_" + fileName);
            FileOutputStream fos = new FileOutputStream(fileDest);

            // 3. Receive Data
            byte[] buffer = new byte[4096];
            int bytesRead;
            long totalRead = 0;
            while (totalRead < fileSize) {
                int remaining = (int)(fileSize - totalRead);
                int toRead = Math.min(buffer.length, remaining);
                bytesRead = dis.read(buffer, 0, toRead);
                if (bytesRead == -1) break;
                fos.write(buffer, 0, bytesRead);
                totalRead += bytesRead;
            }
            fos.close();

            // 4. Feedback
            dos.writeUTF("Success! Received " + totalRead + " bytes.");

            dis.close(); dos.close(); s.close(); ss.close();
        } catch (IOException i) { i.printStackTrace(); }
    }
}
```

Listing 1: Server.java

4.2 Client Implementation

The client sends the metadata first, then streams the actual file content.

```java
package practice_1;
import java.net.*;
import java.io.*;

public class Client {
    public static void main(String[] args) {
        try {
            Socket client = new Socket("localhost", 5000);
            File file = new File("D:\\test_socket.txt");

            if (!file.exists()) return;

```

```
            DataOutputStream dos = new DataOutputStream(client.getOutputStream());
            DataInputStream dis = new DataInputStream(client.getInputStream());
            FileInputStream fis = new FileInputStream(file);

            // 1. Send Metadata
            dos.writeUTF(file.getName());
            dos.writeLong(file.length());

            // 2. Send Content
            byte[] buffer = new byte[4096];
            int bytesRead;
            while ((bytesRead = fis.read(buffer)) != -1) {
                dos.write(buffer, 0, bytesRead);
            }
            dos.flush();

            // 3. Receive Feedback
            System.out.println("Server: " + dis.readUTF());

            fis.close(); dos.close(); dis.close(); client.close();
        } catch (IOException e) { e.printStackTrace(); }
    }
}
```

Listing 2: Client.java

5 Roles and Responsibilities

Server : Listens on port 5000, interprets metadata, and saves incoming bytes to disk.

Client : Connects to the server, validates file existence, and streams data securely.