Practical Work 1: File Transfer over TCP/IP

Nguyen Thi Minh Nguyet
ID: 22BA13241
Class: CS

November 26, 2025

1 Introduction

The goal of this practical work is to design and implement a one-to-one file transfer system using the TCP/IP protocol. The system consists of a **Client** (sender) and a **Server** (receiver). The implementation demonstrates the use of core socket API functions such as `socket()`, `bind()`, `listen()`, `connect()`, and advanced options like `setsockopt()`.

2 Protocol Design

Since TCP is a stream-oriented protocol, it does not preserve message boundaries. To ensure the file is transferred correctly, we designed a simple application-layer protocol involving a metadata handshake.

1. **Metadata:** The client sends the filename and file size (e.g., `"data.txt|1024"`).
2. **Acknowledgement:** The server receives the metadata and replies with an `ACK`.
3. **Data Stream:** The client sends the binary content of the file.

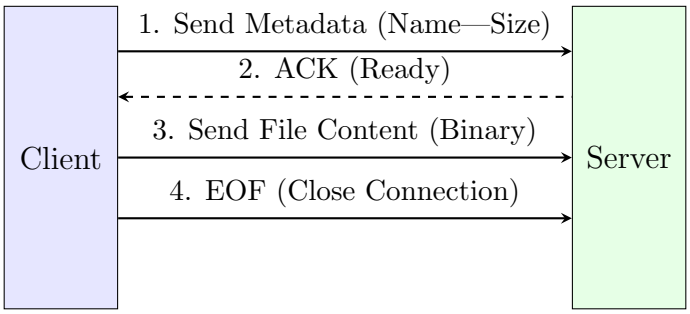

Figure 1: File Transfer Protocol Sequence

3 System Organization

The system architecture follows the standard Client-Server model. The diagram below illustrates the flow of socket API calls required to establish the connection and transfer data.

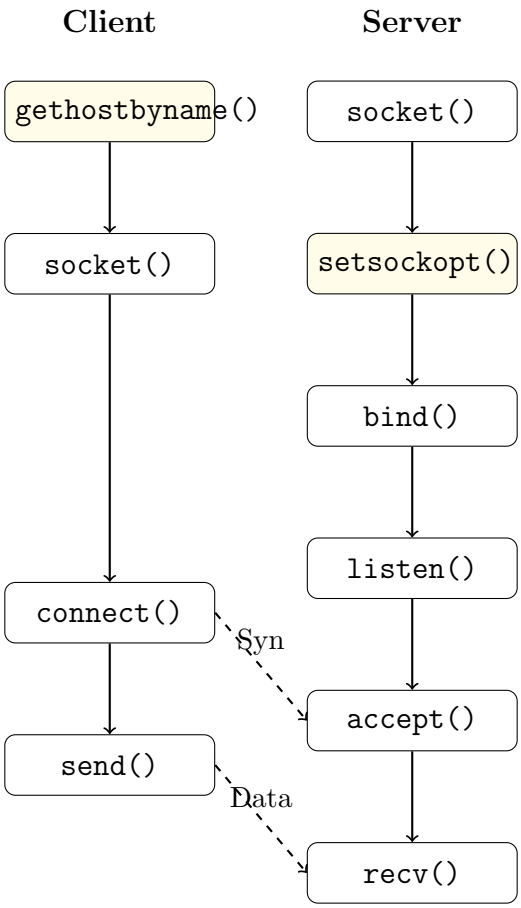

Figure 2: System Organization and Socket API Flow

4 Implementation Details

The implementation adheres to the requirements by utilizing the specific socket functions identified in the course materials.

4.1 Server Implementation

The server uses `setsockopt` to allow address reuse and standard connection handling.

```
import socket, os

def start_server():
    # 1. Create Socket
    server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

    # 2. Set Options (Avoid "Address already in use")
    server.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
```

```
    # 3. Bind and Listen
    server.bind(("127.0.0.1", 65432))
    server.listen()
    print("[LISTENING] Waiting for connection...")

    while True:
        # 4. Accept Connection
        conn, addr = server.accept()

        # 5. Receive Metadata
        data = conn.recv(1024).decode("utf-8")
        filename, filesize = data.split("|")

        # 6. Send ACK and Receive File
        conn.send(b"ACK")
        # ... (Loop to receive file content) ...

        conn.close()
```

Listing 1: Server Code Snippet

4.2 Client Implementation

The client uses `gethostbyname` to resolve the server address before connecting.

```
import socket, os

def start_client():
    # 1. Resolve Hostname
    server_name = "localhost"
    server_ip = socket.gethostbyname(server_name)

    # 2. Create Socket and Connect
    client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    client.connect((server_ip, 65432))

    # 3. Send Metadata
    filename = "image.png"
    size = os.path.getsize(filename)
    client.send(f"{filename}|{size}".encode("utf-8"))

    # 4. Wait for ACK and Send File
    msg = client.recv(1024)
    if msg == b"ACK":
        with open(filename, "rb") as f:
            # ... (Loop to send chunks) ...
            client.send(f.read(1024))

    client.close()
```

Listing 2: Client Code Snippet

5 Conclusion

The file transfer system was successfully implemented. The use of `setsockopt` improved server stability during restarts, and `gethostbyname` added flexibility to the client's con-

nection logic. The protocol design ensures that files are reconstructed correctly on the server side.