Practical Work 1: TCP File Transfer Report

Nghiem Xuan Son- 23BI14388

November 2025

1 Protocol Design

To ensure the server correctly identifies the file name before receiving the raw binary content, we designed a simple application-level protocol using a delimiter.

The communication flow is as follows:

1. The client establishes a TCP connection to the server.
2. The client sends a header containing the filename followed by a newline character (`\n`). This character acts as a delimiter.
3. Immediately after the delimiter, the client streams the raw binary content of the file until EOF (End of File).

Data structure sent by Client:

```
[Filename bytes] + [0x0A (\n)] + [File Content Binary Data...]
```

The server reads the incoming stream, buffers data until it finds the first `\n`. The data preceding it is extracted as the filename, and all subsequent data is treated as file content and written to disk.

2 System Organization

The system follows a standard Client-Server architecture using blocking TCP sockets.

- **Server:** The server runs passively. It binds to a specific port (e.g., 12345) and listens for incoming requests. Upon accepting a client connection, it enters a receiving loop to process the protocol header and then writes the received data chunks to a new file on the disk. It closes the connection upon receiving EOF.
- **Client:** The client runs actively. It requires the IP address and port of the server. It reads a local file, connects to the server, sends the protocol header, and then streams the file content in defined chunk sizes (e.g., 4096 bytes) before closing the connection.

3 Implementation

The implementation uses Python's built-in `socket` library. Files are opened in binary mode (`'rb'` for read binary, `'wb'` for write binary) to handle all file types correctly.

3.1 Client-Side Implementation (Sending)

The following snippet shows how the client prepares the header and sends the file content in chunks.

```python
# Prepare header (filename + newline)
filename = os.path.basename(filepath)
header = filename.encode('utf-8') + b'\n'
client_socket.sendall(header)

# Open file in binary mode and stream content
with open(filepath, 'rb') as f:
    while True:
        # Read chunk from disk
        data = f.read(BUFFER_SIZE)
        if not data:
            break # EOF reached
        # Send chunk over socket
        client_socket.sendall(data)
```

Listing 1: Client sending mechanism

3.2 Server-Side Implementation (Receiving)

The following snippet demonstrates how the server separates the header from the body using the newline delimiter.

```python
received_data = b''
# Loop to find the delimiter
while True:
    chunk = client_socket.recv(BUFFER_SIZE)
    received_data += chunk
    # Check for delimiter
    if b'\n' in received_data:
        # Split data: before \n is filename, after is content start
        header, file_content_start = received_data.split(b'\n', 1)
        filename = header.decode('utf-8').strip()
        break
```

Listing 2: Server header parsing mechanism

4 Who does what

Define the roles and responsibilities of the group members below.

Name	Responsibilities
Son Nghiem	Protocol design, Server implementation, Report writing
Son Nghiem	Client implementation, Testing and debugging
Son Nghiem	System architecture design, Documentation review