Practical Work 1: TCP File Transfer System Report

Nguyen Tuan Dung
Student ID: 23BI14113

November 22, 2025

1 Introduction

This report describes the design and implementation of a simple TCP-based file transfer system consisting of a Client and a Server. The system uses a custom lightweight protocol that defines how metadata (filename and file size) and file content are transmitted reliably over a TCP stream.

2 Protocol Design

2.1 Objectives

The protocol is designed to:

- Reliably transfer arbitrary files using TCP.
- Maintain simplicity and clarity.
- Ensure the server knows exactly what and how much to read.
- Allow future extension (e.g., multiple files, integrity checks).

2.2 Protocol Format

The client sends data to the server in this exact order:

1. **8-byte metadata header**
   - 4 bytes: filename length (unsigned int, big-endian)
   - 4 bytes: file size in bytes (signed int, big-endian)
2. **Filename bytes** (UTF-8 encoded, length defined above)
3. **File content** sent in fixed-size chunks until all bytes are transmitted.

2.3 Protocol Message Layout

```
+----------------------+------------------------+------------------------+
| 4 bytes (uint32)     | 4 bytes (int32)        | N bytes (filename)     |
| Filename Length      | File Size              | UTF-8 Filename         |
+----------------------+------------------------+------------------------+
|                       Binary File Content (chunked)                    |
+----------------------------------------------------------------------+
```

Figure 1: Protocol Message Structure

3 System Organization

3.1 Components

The system consists of:

- **Client:** Reads a local file, sends metadata, and streams file content.
- **Server:** Accepts connections, receives metadata, and reconstructs the file.

3.2 Architecture Diagram

```
+-----------+                                   +-----------+
|           | ---(metadata: length,size)------> |           |
|  Client   | ---(filename)-------------------> |  Server   |
|           | ---(file chunks)----------------> |           |
+-----------+                                   +-----------+
```

Figure 2: Client-Server Architecture

4 Implementation Summary

4.1 Client: Sending Metadata

Listing 1: Client sending metadata

```
filename_len = len(filename)
filesize = os.path.getsize(filepath)

header = struct.pack('>Ii', filename_len, filesize)
s.sendall(header)
s.sendall(filename.encode('utf-8'))
```

4.2 Client: Streaming File Content

Listing 2: Client sending file chunks

```
with open(filepath, ’rb’) as f:
   while True:
      chunk = f.read(BUFFER_SIZE)
      if not chunk:
         break
      s.sendall(chunk)
```

4.3 Server: Receiving Metadata

Listing 3: Server receiving metadata

```
raw_header = conn.recv(8)
filename_len, filesize = struct.unpack(’>Ii’, raw_header)

filename = conn.recv(filename_len).decode(’utf-8’)
```

4.4 Server: Writing File Data

Listing 4: Server writing file chunks

```
with open("RECEIVED_" + filename, ’wb’) as f:
   bytes_received = 0
   while bytes_received < filesize:
      chunk = conn.recv(min(BUFFER_SIZE, filesize - bytes_received))
      if not chunk:
         break
      f.write(chunk)
      bytes_received += len(chunk)
```

5 Conclusion

The implemented TCP file transfer system demonstrates a clear and robust method for transmitting files from a client to a server using a custom protocol. By sending metadata first, the server always knows how to safely reconstruct the incoming file. This structure also allows easy future extensions such as authentication, multi-file transfers, or checksum verification.