Practical Work 1: TCP File Transfer

Phạm Tưởng Ngạn
Distributed Systems

November 21, 2025

1 Protocol Design

To ensure reliable file transfer over the TCP stream, I designed a simple application-layer protocol. Since TCP is a stream-oriented protocol, it does not inherently respect message boundaries. Therefore, my protocol explicitly defines the metadata and content boundaries to avoid packet coalescing (often called the ”sticky packet” problem) or partial reads.

The protocol structure consists of a header followed by the file payload. The header contains the filename size, the actual filename, and the file size. This allows the receiver to know exactly how many bytes to read for each stage of the transfer.

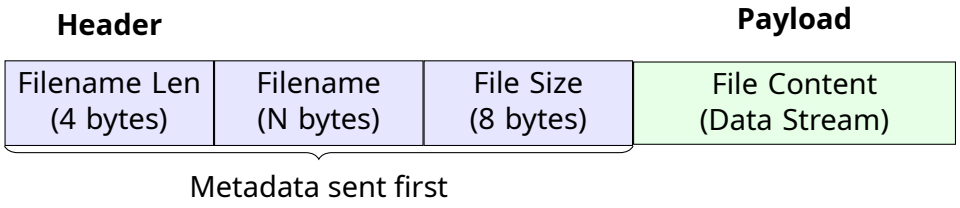

Figure 1: Application Layer Protocol for File Transfer

1.1 Field Description

- **Filename Length:** An integer indicating the length of the string that follows.
- **Filename:** The actual name of the file (useful for saving on the server side).
- **File Size:** A long integer indicating the total size of the file in bytes.
- **Content:** The binary data of the file, read in chunks until *File Size* is reached.

2 System Organization

I organized the system into a Client-Server architecture using standard Blocking I/O sockets. The server acts as a passive entity that binds to a specific port and listens for incoming connections, while the client acts as the active entity that initiates the connection.

As illustrated in Figure 2:

1. The **Server** initializes a socket, binds it to an address, and enters a listening state.
2. The **Client** creates a socket and connects to the server’s address.

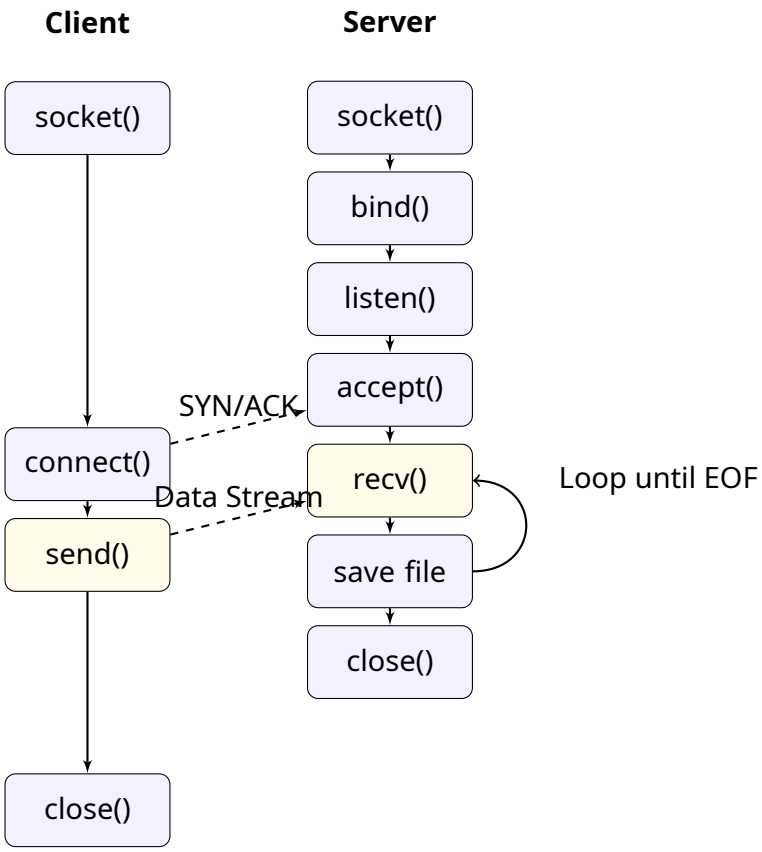

Figure 2: System Architecture and Socket Flow

3. Once the connection is established (via the Three-way handshake), the Client begins the file transfer protocol described in Section 1.
4. The Server reads the metadata first, opens a file pointer, and then loops to write the incoming chunks to disk.

3 Implementation

My implementation handles the reading and writing of files in binary mode (‘rb‘ and ‘wb‘) to support all file types (images, text, executables). Below are the key snippets for the transfer logic.

3.1 Client Side: Sending the File

The client is responsible for packing the header and streaming the content.

```
import socket
import os
import struct

def send_file(sock, filename):
    # 1. Get file stats
    filesize = os.path.getsize(filename)

    # 2. Create Header: filename_len (4 bytes), filename, filesize (8 bytes)
    # '>I' = big-endian unsigned int, '>Q' = big-endian unsigned long long
    filename_encoded = filename.encode('utf-8')
```

```
    header = struct.pack('>I', len(filename_encoded)) + \
             filename_encoded + \
             struct.pack('>Q', filesize)

    sock.sendall(header)

    # 3. Send File Content in Chunks
    with open(filename, 'rb') as f:
        while True:
            chunk = f.read(4096) # 4KB Buffer
            if not chunk:
                break
            sock.sendall(chunk)
    print("File transfer completed.")
```

Listing 1: Client Implementation (Python)

3.2 Server Side: Receiving the File

The server uses a buffered approach to receive the exact amount of data specified in the header.

```
import socket
import struct

def recv_file(sock):
    # 1. Read Filename Length (4 bytes)
    raw_len = sock.recv(4)
    if not raw_len: return
    filename_len = struct.unpack('>I', raw_len)[0]

    # 2. Read Filename
    filename = sock.recv(filename_len).decode('utf-8')

    # 3. Read File Size (8 bytes)
    raw_size = sock.recv(8)
    filesize = struct.unpack('>Q', raw_size)[0]

    # 4. Read Content Loop
    received = 0
    with open(f"received_{filename}", 'wb') as f:
        while received < filesize:
            # Read remaining bytes or standard buffer size
            bytes_to_read = min(4096, filesize - received)
            chunk = sock.recv(bytes_to_read)
            if not chunk: break
            f.write(chunk)
            received += len(chunk)

    print(f"Received {filename} ({received} bytes)")
```

Listing 2: Server Implementation (Python)