Report: RPC File Transfer System

Student Name: [Nguyen The Thinh]
Student ID: [22BA13295]

1 Design of the RPC Service

To implement a robust and efficient file transfer service, we adopted a **Hybrid Protocol** approach. This protocol combines the structured clarity of JSON-RPC for control messages with the raw performance of TCP streams for data transfer.

The communication process consists of two distinct phases:

- **Phase 1 (Handshake):** The Client sends a JSON packet containing metadata (method name, filename). The Server parses this JSON and responds with a confirmation ("OK").
- **Phase 2 (Data Stream):** Upon receiving confirmation, the Client switches to raw binary mode and streams the file content directly. The Server reads until the stream ends (EOF).

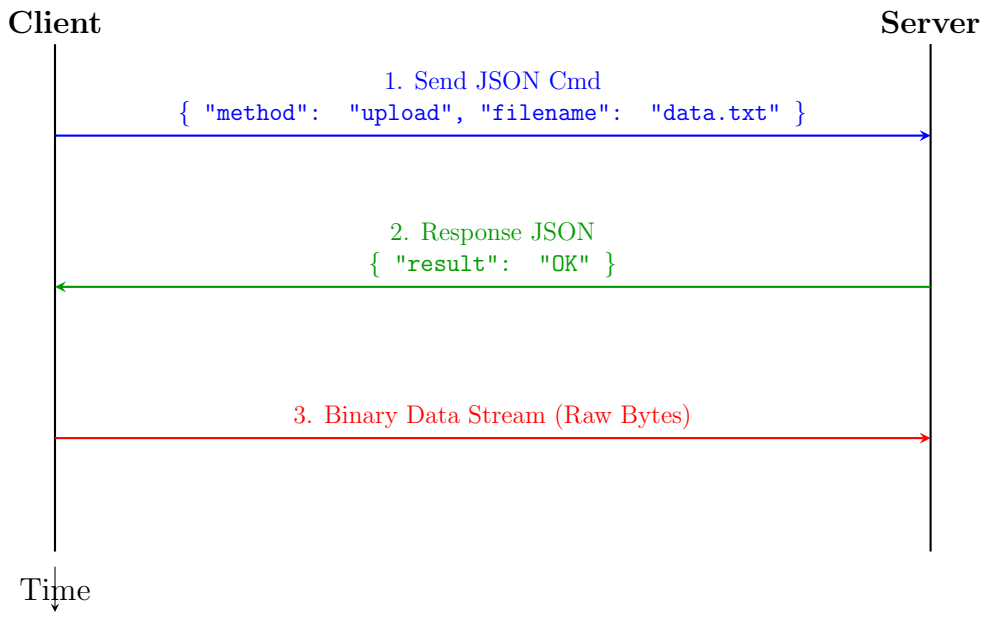

Figure 1: Sequence Diagram of the RPC File Transfer Protocol

2 System Organization

The system follows a classic **Client-Server Architecture**. It is organized into two main standalone applications written in C++:

- **Client Node:** Responsible for reading local files, constructing JSON payloads, and initiating the connection. It handles user input (CLI arguments) to determine the target server and file.

- **Server Node:** A persistent listener that waits for incoming connections on a specific port (2702). It handles JSON parsing and file writing operations.

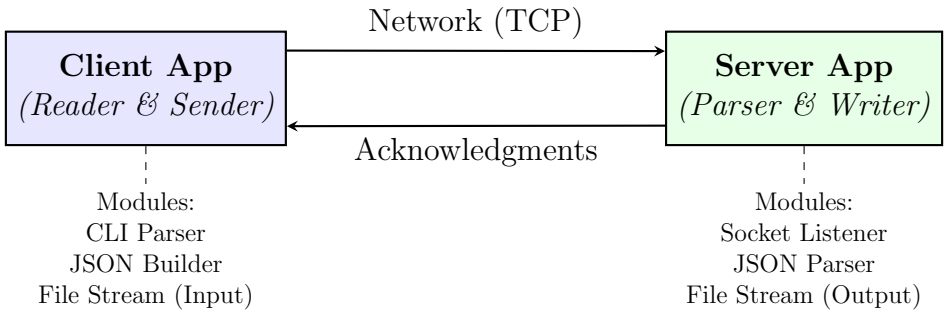

Figure 2: System Organization and Component View

3 Implementation of File Transfer

The implementation leverages C++ Standard Library features such as `std::string` for JSON manipulation and `std::fstream` for binary file operations.

Below is the core snippet showing the transition from JSON parsing to Binary file writing on the Server side. This demonstrates how we extract the filename and efficiently save the incoming data stream.

```
// 1. Read the JSON Command first
int n = read(newsockfd, buffer.data(), buffer.size());
string json_str(buffer.data());

// 2. Parse JSON to get filename (Custom Helper Function)
string method = parse_json_value(json_str, "method");
if (method == "upload") {
    string filename = parse_json_value(json_str, "filename");

    // 3. Send Acknowledgment
    string response = "{\"result\": \"OK\"}";
    write(newsockfd, response.c_str(), response.length());

    // 4. Switch to Binary Mode for File Writing
    ofstream outfile(filename, ios::binary);

    // Loop to receive raw bytes until EOF
    while (true) {
        int bytes_received = read(newsockfd, buffer.data(), buffer.size());
        if (bytes_received <= 0) break; // End of stream

        outfile.write(buffer.data(), bytes_received);
    }
    outfile.close();
}
```

Listing 1: Server-side Implementation: JSON Parsing and Binary Writing

This logic ensures that large files are transferred in chunks without loading the entire file into memory, providing efficiency and scalability.