TCP File Transfer Protocol Design

Pham Tung Anh

November 29, 2024

1 Protocol Design

The TCP file transfer protocol is designed to ensure reliable and efficient transmission of files between a client and a server using sockets. The design leverages TCP's connection-oriented nature to guarantee delivery and proper sequencing of data packets.

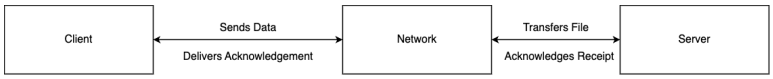

Figure 1: Protocol Design for File Transfer

2 System Organization

The system consists of two main components:

- **Client:** Reads the file and sends it in chunks to the server.
- **Server:** Receives the file and writes the received data to disk.

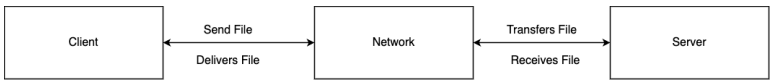

Figure 2: System Organization

3 Implementation

Below is the implementation of the client-side and server-side code in C.

3.1 Client-Side Code

The client creates a socket, connects to the server, and sends file data in chunks.

Listing 1: Client-Side Code

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <arpa/inet.h>

#define PORT 8080
#define BUFFER_SIZE 1024

int main() {
    int sock = 0;
    struct sockaddr_in serv_addr;
    char buffer[BUFFER_SIZE] = {0};

    if ((sock = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
        printf("\n Socket creation error \n");
        return -1;
    }

    serv_addr.sin_family = AF_INET;
    serv_addr.sin_port = htons(PORT);

    if (inet_pton(AF_INET, "127.0.0.1", &serv_addr.
       sin_addr) <= 0) {
        printf("\nInvalid address/ Address not supported 
           \n");
        return -1;
    }

    if (connect(sock, (struct sockaddr *)&serv_addr,
       sizeof(serv_addr)) < 0) {
        printf("\nConnection Failed \n");
        return -1;
    }

    FILE *file_to_send = fopen("file_to_send.txt", "r");
    if (file_to_send == NULL) {
        perror("File open error");
        exit(EXIT_FAILURE);
    }

    int bytes_read;
    while ((bytes_read = fread(buffer, 1, BUFFER_SIZE,
       file_to_send)) > 0) {
```

```
        send(sock, buffer, bytes_read, 0);
    }

    fclose(file_to_send);
    printf("File␣sent␣successfully.\n");

    return 0;
}
```

3.2 Server-Side Code

The server listens for connections, accepts the client's connection, and writes received data to a file.

Listing 2: Server-Side Code

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <arpa/inet.h>

#define PORT 8080
#define BUFFER_SIZE 1024

int main() {
    int server_fd, new_socket;
    struct sockaddr_in address;
    int addrlen = sizeof(address);
    char buffer[BUFFER_SIZE] = {0};

    if ((server_fd = socket(AF_INET, SOCK_STREAM, 0)) ==
       0) {
        perror("socket␣failed");
        exit(EXIT_FAILURE);
    }

    address.sin_family = AF_INET;
    address.sin_addr.s_addr = INADDR_ANY;
    address.sin_port = htons(PORT);

    if (bind(server_fd, (struct sockaddr *)&address,
       sizeof(address)) < 0) {
        perror("bind␣failed");
        exit(EXIT_FAILURE);
    }

    if (listen(server_fd, 3) < 0) {
        perror("listen");
```

```
        exit(EXIT_FAILURE);
    }

    if ((new_socket = accept(server_fd, (struct sockaddr
       *)&address, (socklen_t*)&addrlen)) < 0) {
        perror("accept");
        exit(EXIT_FAILURE);
    }

    FILE *received_file = fopen("received_file.txt", "w")
       ;
    if (received_file == NULL) {
        perror("File open error");
        exit(EXIT_FAILURE);
    }

    int bytes_received;
    while ((bytes_received = recv(new_socket, buffer,
       BUFFER_SIZE, 0)) > 0) {
        fwrite(buffer, 1, bytes_received, received_file);
    }

    fclose(received_file);
    printf("File received successfully.\n");
    close(new_socket);
    close(server_fd);

    return 0;
}
```

4 Responsibilities

- **Client:** Initiates the connection, reads the file, and sends data chunks to the server.
- **Server:** Accepts the connection, receives data, and saves the file.