

TCP File Transfer System: Architecture and Implementation

Technical Documentation

November 25, 2024

1 Introduction

The system implements a client-server architecture for file transfer over TCP/IP. It consists of three main components:

$$\text{Components} = \{\text{Client}, \text{Server}, \text{Script Interface}\}$$

2 System Architecture

The system follows a traditional client-server architecture with the following mathematical representation:

S = Server Process

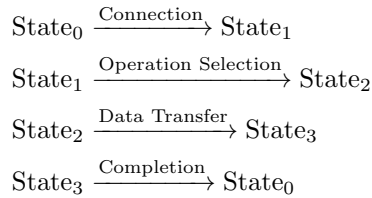
C = Client Process

F = File Data

P = Protocol Operations = {SEND, REQUEST, LIST}

2.1 Communication Protocol

The protocol follows a state machine model where each operation $p \in P$ transitions through states:



3 Key Components

3.1 FileTransferServer

The server implements a listening socket with the following configuration:

$$\text{Server Socket} = \begin{cases} \text{Host:} & 0.0.0.0 \\ \text{Port:} & 12345 \\ \text{Backlog:} & 1 \\ \text{Protocol:} & \text{TCP} \end{cases}$$

3.2 FileTransferClient

The client implements three main operations:

$$\text{Client Operations} = \begin{cases} \text{send_file}(f) & : \text{Upload file } f \text{ to server} \\ \text{request_file}(f) & : \text{Download file } f \text{ from server} \\ \text{list_files}() & : \text{Get server file listing} \end{cases}$$

3.3 Data Transfer Protocol

For file transfers, the protocol follows this sequence:

- Step 1 : Initialize connection
- Step 2 : Send operation type $p \in P$
- Step 3 : Exchange metadata (filename, size)
- Step 4 : Transfer data in chunks of 4096 bytes
- Step 5 : Verify completion

The progress of file transfer is calculated as:

$$\text{Progress}(\%) = \frac{\text{Bytes Transferred}}{\text{Total Bytes}} \times 100$$

4 Implementation Details

4.1 Buffer Sizes

The system uses optimized buffer sizes:

Control Messages : 1024 bytes
Data Transfer : 4096 bytes

4.2 Error Handling

The system implements comprehensive error handling with the following categories:

$$\text{Errors} = \begin{cases} \text{Connection Failures} \\ \text{File Not Found} \\ \text{Transfer Interruption} \\ \text{Invalid Operations} \end{cases}$$

5 Usage Examples

5.1 Server Mode

To start the server:

```
python script.py --mode server --port 12345
```

5.2 Client Mode

For client operations:

```
Send : python script.py --mode client --sor send --fn file.txt
Receive : python script.py --mode client --sor receive --fn file.txt
List : python script.py --mode client --sor list
```

6 Performance Characteristics

The system's performance can be characterized by:

$$\text{Transfer Time} = \frac{\text{File Size}}{\text{Network Bandwidth}} + \text{Protocol Overhead}$$

Where protocol overhead includes:

$$\text{Overhead} = \text{Connection Setup} + \text{Metadata Exchange} + \text{Acknowledgments}$$

7 Code Structure

7.1 Class Hierarchy

The system is organized into two main classes with their interfaces:

FileTransferServer $\rightarrow \{\text{run, handle_file_transfer, send_file, receive_file}\}$
FileTransferClient $\rightarrow \{\text{send_file, request_file, list_files}\}$

7.2 Protocol Flow

The protocol flow can be represented as a sequence:

