Department of Computer Science

CPSC 471: Computer Communications

TCP Client-Server File Transfer

Nhi Danis, Junhao Guo, Alvaro Samayoa, Peter West, Steven Delgado

Prof. M. Kurban

May 03, 2024

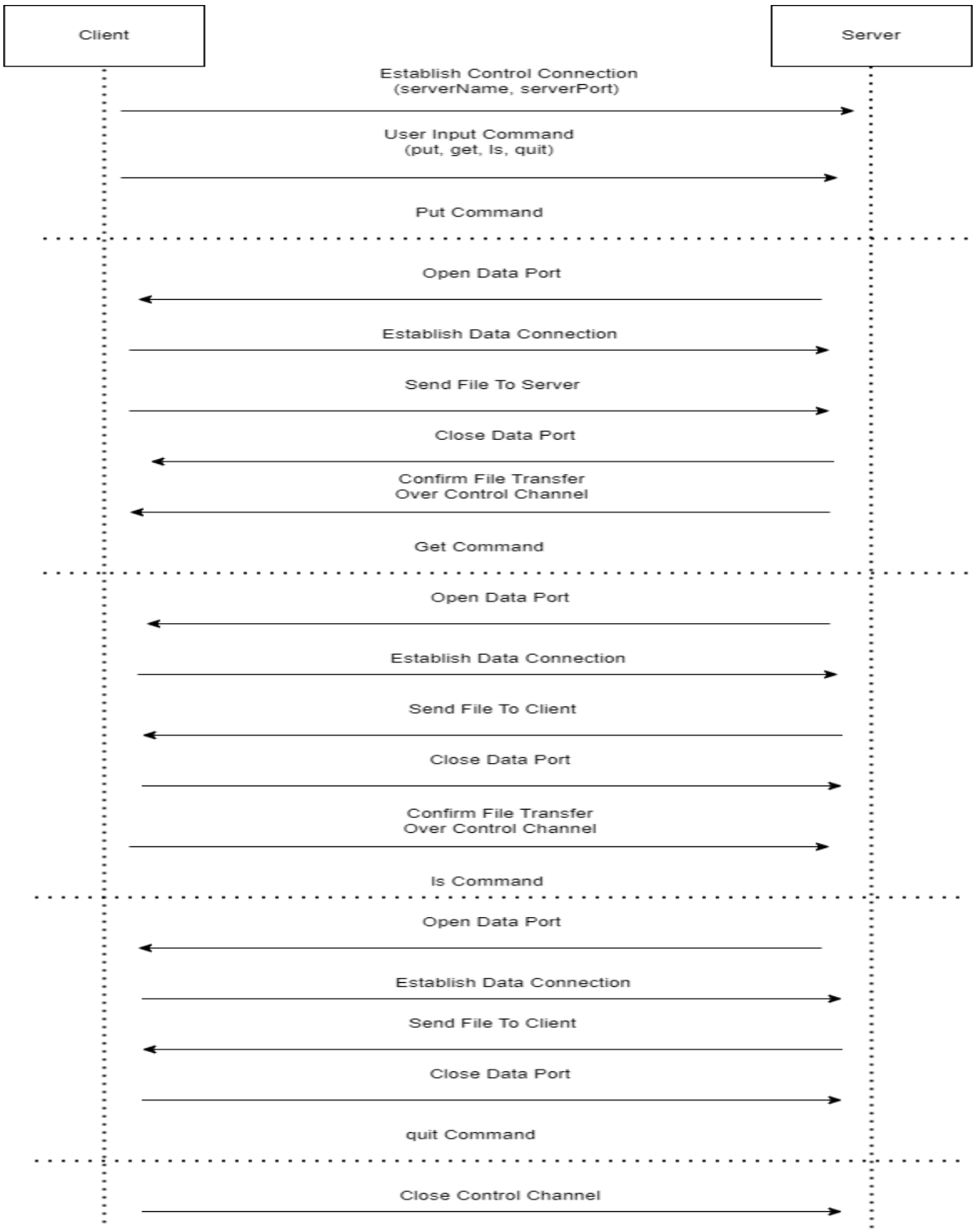
Table of contents

Protocol Design

Our TCP client-server file transfer protocol is designed to be simple and efficient. Clients interact with the server by sending commands, which consist of straightforward instructions to perform various actions. These actions include uploading files to the server (put <filename>), downloading files from the server (get <filename>), listing files available on the server (ls), and disconnecting from the server (quit). For commands like put and get, clients provide additional information, such as filenames, to specify the files being transferred. Communication between the client and server occurs through two channels: the control channel and the data channel. The control channel facilitates the exchange of commands and responses, allowing clients to send commands to the server and receive responses accordingly. Meanwhile, the data channel manages the actual transfer of file data, ensuring that files are transferred securely and efficiently. Error handling is also an integral part of our protocol. Both the client and server perform integrity checks to verify data integrity before processing commands or transferring files. In case of errors, such as invalid commands or files not found, the server sends error messages to the client to notify them of the issue. Additionally, when a client decides to disconnect from the server (quit command), both sides gracefully terminate the connection, ensuring that resources are released properly.

Protocol Diagram

This diagram illustrates the flow of communication between client and server during command exchange and data transfer.



Example of Running the Program

To operate the server side, first, open a terminal window. Then, navigate to the directory where the server.py file is located using the cd command, for example: cd /path/to/server/directory. Once in the correct directory, initiate the server by running the command python server.py <port>, replacing <port> with the desired port number (e.g., python server.py 12345).

On the client side, open another terminal window. Navigate to the directory containing the client.py file using the cd command, similar to the server side. Start the client by executing the command python client.py <server_address> <port>. Replace <server_address> with the IP address or hostname of the server (e.g., localhost or 192.168.1.100) and <port> with the port number on which the server is listening (e.g., 12345).

Once the client-server connection is established, users can execute various commands to interact with the server. For instance, the put <filename> command allows users to upload a file to the server by specifying the filename, while get <filename> enables them to download a file from the server. Additionally, users can use the ls command to list files available on the server or the quit command to gracefully disconnect from the server.

Modules Use

The socket module serves the purpose of creating TCP/IP sockets, facilitating communication between the client and server. These sockets act as endpoints for data transmission, allowing for the exchange of information over the network.

Threading, another crucial module, enables concurrent handling of multiple client connections on the server. By utilizing threads, the server can manage multiple client requests simultaneously, improving responsiveness and efficiency.

Additionally, the OS module provides essential file operation functionalities, such as listing directory contents. This module allows the server to access and manipulate files on the file system, enabling tasks like serving files to clients or performing file management operations.

The sys module was used to access command line arguments. This is how certain variable like port number and address were passed to the client and server.

The cmd module was used for building the command-line interpreter application. This was used by the client for communication to the server.

server.py is the main module for running the server application. It starts the server and then listens for an incoming connection on a specified port.

client.py is the main module for running the client application. It looks to connect with a server given an address and port. It then handles input commands such as ls, put, get and quit.

helper.py is the module primarily used for handling the send and receive logic used by both the server and client.