CP372 Computer Networks
Winter 2025

REPORT

Socket Programming
Client-Server Communication Application

Hargun Kaur - 169023254
Rishubh Gusain - 169044443

February 10, 2025

CP372 Computer Networks
Winter 2025

A brief description of the code, outlining the steps, operation, and any special considerations

Brief Description

This project uses Python socket programming to create a basic client-server communication application. Multiple clients can connect to a single server since the connection is established using TCP sockets. The server manages communication by giving each client a unique name, keeping a session cache, and limiting the number of concurrent connections.

Client operations:

- Connects to the server by utilizing TCP sockets
- Sends a unique name upon connection.
- Can send messages, request status, and exit.
- Supports file downloading from the server.
- Implements basic error handling.

Server operations:

- The server binds to *IP: 172.20.10.2*, *Port: 12345*
- The hardcoded limit of client connections is 3
- Uses multi-threading to handle multiple clients at once
- Maintains a cache of connected clients, tracking connection times
- Responds with ACK to each client message
- Supports file listing and file transfer (rubric bonus)

Outline of the steps/Operation

1. Setting up the server
    a. After initializing a TCP socket, the server watches for new client connections.
    b. Up to 3 clients can connect at the same time.
    c. Every client is given a unique identification number, i.e. "Client01" or "Client02".
    d. To keep track of client connection information, the server keeps a session cache.
2. Client connection
    a. Clients use a TCP socket to establish a connection with the server.
    b. The client gives the server its allocated name when it connects.
    c. Messages can be sent from the client using the command line.
    d. Received messages are echoed by the server, which adds "ACK" to indicate reception.

3. Features for communication
    a. The server acknowledges messages sent by clients.
    b. Sending "status" to the server will cause it to return a list of clients that are currently connected.
    c. The server deletes the client from the session cache when a client sends the "exit" command to disconnect.
4. File repository - bonus
    a. A file repository is kept up to date by the server.
    b. To obtain a list of the available files, clients can submit "list".
    c. The server will broadcast a certain file to clients upon request.
    d. Errors like asking for a file that doesn't exist are handled by the server.

Any difficulties faced and how they were handled (if any)

Multiple clients multi-threading

Scalability was initially constrained by the server's ability to support only one client at a time. Multi-threading was used to solve this problem, enabling several clients to connect and communicate with the server at once. This guaranteed a more seamless user experience and increased efficiency.

Client disconnections

Managing sudden client disconnections, which interfered with the session cache, was another difficulty. These unforeseen disconnections led to discrepancies if they weren't handled properly. Exception handling was implemented to address this issue and guarantee that disconnected clients were swiftly cleared from the cache, preserving system stability.

File transfer feature

It was challenging to implement a file transfer capability, particularly when it came to handling big files without using up too many resources. Sending files while also making sure that it works correct error handling was a challenge. Chunked file streaming, which divides data into smaller chunks for delivery, was used to avoid this. This method guaranteed dependable data flow across the socket connection while optimizing performance.

Test result: Make sure to test all items in the rubric and provide the necessary screenshot of your testcase outputs in the report.

```
[*] Connected to the server at 172.20.10.2:12345.
Type messages or commands ('status', 'list', 'get <filename>', 'exit').
You: list
Server: Screenshot (1).png
You:
```

Testing criteria:

- Naming client connections —> Clients receive unique names upon connection
- Exchange of messages —> Server sends a message with “ACK”
- Limit on clients —> Clients after 3rd are rejected
- Status —> Server returns details of active clients & displays correct cache info
- Disconnecting —> Disconnected clients are removed from the server swiftly
- File transfer —> Server lists the files which are also downloadable

Possible improvements: If you had given more time, what would you add or do differently?

With more time, several improvements might be made to increase the application's robustness and utility.

1. Increasing the limit of clients - Three concurrent clients are the server's hardcoded maximum at the moment. This limits scalability even as it guarantees controlled resource management. Making the client limit adjustable would be a preferable strategy, enabling the server administrator to dynamically set the maximum number of connections. Configuration files or command-line parameters could be used to accomplish this, increasing the system's adaptability for various use cases.
2. Implementing a GUI (Graphical User Interface) - The CLI (Command-line Interface) used in our current implementation might not be intuitive for each and every user. Adding a GUI to the server and client could increase usability greatly. A graphical chat window with buttons for file transfers and text input would streamline communication and eliminate the need for manual commands. An intuitive interface could be designed using libraries like PyQt or Tkinter.
3. Long-term storage - Currently, the server just keeps client connection information in memory. This implies that all connection history is erased upon server restart. Tracking client behaviour over time would be possible with the implementation of persistent storage, such as logging client session details in a database or text file. Debugging, tracking usage trends, and keeping security records would all benefit from this.
4. Advanced features (MFA, group chatting) - Features like user authentication, which requires each client to enter their login credentials before they can access the server, can improve the chat system. The application would also be more

useful if it included a group chat mode that enables several clients to interact in a common chatroom. Enhancements to file-sharing, like resumable downloads and progress indicators, might be additional benefits.

This project complies with the assignment's requirements and effectively uses Python sockets to construct a simple client-server communication system. The server assigns distinct client names, enables message exchange, and effectively manages several clients through multi-threading. As an added bonus, it allows file listing and transfer and keeps track of open connections in a cache. Simple commands can be used to communicate with the server from the client, facilitating smooth communication. The system can be improved in a number of ways, including enhancing security, adding a graphical user interface, and enabling persistent storage, even if it satisfies the basic networking notions. All things considered, this project shows a useful grasp of socket programming and offers a strong basis for further research and practical applications.

The End