Terminal in Golang via Transmission Control Protocol

Piñarte Diaz Yohan Stevens (160004926), Aristizabal Sabogal Juan Esteban (160004903)

Abstract

This document addresses the logical connection process between a client and a server in a TCP/IP communication environment. In the initial stage of this connection, the server awaits incoming connections from the client, establishing a TCP socket that listens on a specific IP address and port. Once the client connects to the server using the designated IP address and port, the interaction begins. During this initial phase, the server requests the client's user credentials, which include a username and password. These credentials are transmitted to the server for validation. In the final stage of the logical connection, the server verifies the authenticity of the user's credentials by consulting a corresponding database. If the credentials are authentic, the server sends a confirmation message to the client, allowing them to continue using the interactive shell. Once authenticated, the client can send commands to the server through the terminal. The server processes these commands, executes the corresponding operations, and returns the results to the client. The exchange of commands and responses between the client and the server persists until the client decides to terminate the connection. When the client issues the "bye" command, the server concludes the connection with that client and prepares for new connections. The logical connection process comprises the initial setup of communication, user authentication, command execution, and the orderly termination of the connection.

Keywords: Terminal, TCP, socket, golang, client-server.

I. INTRODUCTION

The interconnection of systems through networks is a fundamental component in the modern architecture of information technology. This project focuses on the implementation of a remote terminal that allows users to connect to a server via a TCP/IP network, send commands to the server, and receive and execute the results of these commands. Such a system is crucial for the remote management and administration of servers and other network devices, providing a powerful and flexible tool for system administrators and IT professionals.

In the context of this project, a remote terminal is an interface that allows interaction with the server's operating system from a client machine over a network. This interaction is carried out by sending commands from the client to the server, where these commands are executed, and the results are returned to the client. Implementing such a system involves a series of components and technical considerations, ranging from network connection management to communication security and user authentication.

Importance of Logical Connections in Networks

Logical connections, such as those established by this project, are crucial for the operation of distributed systems and the remote management of IT infrastructures. These connections allow machines to communicate efficiently over networks, facilitating tasks such as server management, file transfers, and remote execution of applications.

In a corporate or data center environment, the ability to manage systems remotely is indispensable. System administrators can perform maintenance tasks, troubleshoot issues, and monitor system performance without needing to be physically present. This not only increases operational efficiency but also reduces the costs associated with the time and resources needed for manual system management.

Project Objectives

The main objective of this project is to design and implement a remote terminal system that allows users to authenticate, send commands, and receive results efficiently and securely. This system should be capable of handling multiple client connections, ensuring that each user can operate independently without interfering with the operations of other users.

Specific Objectives:

1. Establishment of TCP Connections: Implement a TCP server that listens on a specific port, accepts incoming client connections, and manages these connections concurrently.

1

2. User Authentication: Design a secure mechanism for user authentication, including the handling and verification of credentials and allowed IPs.

3. Command Execution: Allow users to send commands to the server, execute these commands in the server environment, and return the results to the client.

4. Error Handling:Implement secure practices for data transmission and error handling, ensuring the integrity and availability of the system.

II. Application Architecture

The project was developed under the Client-Server architecture, which involves dividing it into two main components: the server and the client. The functioning and implementation of each component are detailed below.

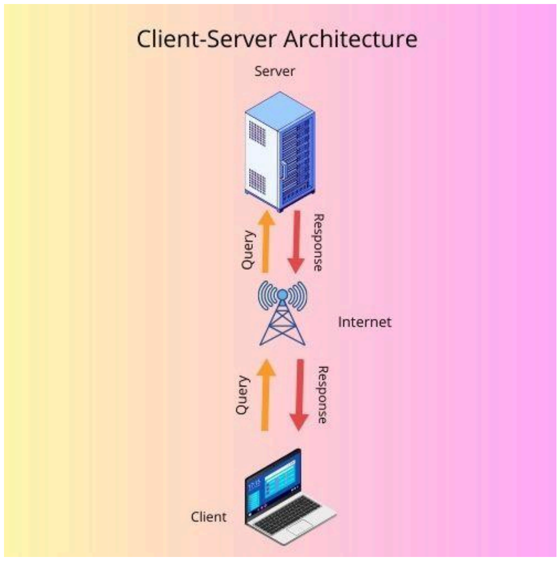

Figure 1. Client-Server Architecture.

Client

The client is responsible for connecting to the server and managing its data flow. The client's structure includes:

Establishment of the TCP Connection:

The client is configured with the transmission control protocol and connects to the server using the specified IP address and port. This connection remains open during the user's session.

Request and Transmission of Credentials:

The client requests the user's username and password, which are then sent to the server. The password is encrypted using the SHA-256 algorithm before being transmitted for security reasons.

Command Sending:

Once authenticated, the client allows the user to enter commands, which are sent to the server for execution. The output of these commands is received and displayed to the user.

Periodic Reports:

At specific intervals provided by the user during execution, the client receives a report on CPU, memory, and disk usage of the server's system.

Server

The server is responsible for accepting client connections, authenticating users, and executing commands. The server's structure includes:

Establishment of the TCP Connection:

The server is configured with the transmission control protocol to listen on a specific port, accepting client connections. Once a connection is established, the server must manage the connections, which can be achieved using threads or "goroutines" in Go.

User Authentication:

Authentication is performed by verifying the credentials sent by the client against a user database file. This file contains securely stored usernames and passwords. The server receives the client's credentials, compares them with the stored ones, and returns the authentication result to the client. Similarly, it verifies that the IP requesting the connection is authorized.

Command Execution:

Once authenticated, the server receives commands from the client. These commands are executed in the server's environment using the assigned UNIX system shell, and the results are sent back to the client. This is done using Go's `os/exec` library, which allows executing operating system commands and capturing their output.

Periodic Reports:

At specific intervals provided by the client, the server sends to this a report on CPU, memory, and disk usage of the system.

III. Execution Environment

Initial Stage of Logical Connection

The server and client are on different machines and operative systems with the server awaiting incoming connections. On the server side has an UNIX operative system, a TCP socket is established, listening on a specific IP address and port. Once listening is set up, the server is ready to accept incoming connections.

The client which has a Windows operative system, connects to the server using the server's specific IP address and port. Once the client connects to the server, communication begins.

The authentication process starts when the client connects to the server. The server requests user credentials from the client, including a username and password. These credentials are sent to the server for verification.

Final Stage of Logical Connection

After the client sends the user credentials to the server, the server verifies their authenticity by checking a user database. If the credentials are valid, the server sends a confirmation message ("1") to the client, allowing the user to proceed with the interactive shell, otherwise (“0”) is sent.

Once authenticated, the client can send commands to the server via the interactive shell. The server processes these commands, executes the corresponding operations, and sends the results back to the client.

The exchange of commands and responses between the client and server continues until the client decides to exit. When the client sends the "bye" command to the server, the server terminates the connection with that specific client and waits for new connections.

In summary, the initial stage of the logical connection involves setting up communication between the client and server, while the final stage involves user authentication, system command execution, and orderly termination of the connection once the client decides to exit.

IV. Conclusion

This project on logical connections through a remote terminal provides a valuable tool for remote system administration. By implementing a server and client that can authenticate and execute commands securely, it offers an efficient solution for managing distributed IT infrastructures. Through proper handling of TCP connections, secure user authentication, and remote command execution, this project demonstrates the importance and potential of system interconnection in modern networks. The implementation of secure practices and efficient error handling ensures that the system is robust and reliable, meeting the necessary operational and security requirements in corporate and data center environments.

References

[1] Standard library - Go Packages. (s. f.). https://pkg.go.dev/std

[2] Rivo. (s. f.). GitHub - rivo/tview: Terminal UI library with rich, interactive widgets — written in Golang. GitHub. https://github.com/rivo/tview

[3] Facundo Carballo. (2023, 16 diciembre). Web Sockets - Guía completa - Golang | React | NextJS [Vídeo]. YouTube. https://www.youtube.com/watch?v=3LNIXVKGezo

[4] Facundo Carballo. (2023a, diciembre 14). Identificar Clientes conectados a un Web Socket - Golang | React | NextJS [Vídeo]. YouTube. https://www.youtube.com/watch?v=5YFa2Mv_dOc