Documentation for Traffic Monitoring Server-Client System

Computer Networks 2023-2024

Romanescu Adia-Ioana (2A3)

January 18^{th}, 2024

1. Introduction

1.1 Overview

This application provides a multi-concurrent server-client system that simultaneously monitors the traffic data of each client and responds accordingly (such as speed, location, and subscriptions). The monitoring will be carried out through the following communication model between the server and clients:

a. Clients will submit orders that will represent:

1. Authentication (registration, login/logout
2. Request for information
   (a) Subscribe to an accessible peco station in that area, a sports channel, or a weather station
   (b) Requesting information about preferences subscribed to
3. Reporting road incidents

b. The server will receive the commands and, depending on the client's authorization, process them to transmit:

1. For each authentication request
   (a) Account sign-in/out status or account registration
2. For each request to report a traffic incident, it will broadcast that incident to all connected clients
3. For every request for information, an appropriate answer
4. As long as the client is authorized (is logged in): the speed at which he drives, the location, and the permitted speed on the respective road segment with a frequency of 1 minute

1.2 Objectives

1. Use of TCP sockets to ensure a reliable connection between the server and the client
2. Realize a multithreaded concurrent server design
3. The use of data structures that allow the transmission of asynchronous messages
4. The app's functionalities should be treated mostly just by the server

2. Applied Technologies

a. TCP/IP

Considering an application with the given purpose, but implemented at an industrial level, we understand the importance of the efficiency required to have communication with the shortest possible time between the server and the clients and to increase the number of clients that can interact with the server so that the responses to the requests are addressed promptly. On the other hand, also from the nature of such an application, it follows that the importance of the integrity of the transmitted data is greater, thus we choose to use the TCP protocol in favor of the UDP one. Because TCP guarantees that data sent from one end will be received accurately and in the correct order by the other end. If any packets are lost or arrive out of order, TCP will retransmit them until the data is correctly received.[6]

b. Sockets

In computer network programming, the application employs the BSD Sockets (POSIX Sockets) API, a tool for handling communication protocols. In this application, the creation of a new communication endpoint involves the socket() primitive, specifying the TCP protocol with domain and type parameters set to AF INET and SOCK STREAM[1]. The bind() primitive assigns an IP address and Port to the socket. Furthermore, the sockets enable data exchange via familiar system calls like write() and read() or primitives such as recvmsg() and sendmsg().[2]

c. Multithreading

A potential approach to implementing client concurrency, along with simultaneous tasks, involves using the fork() primitive to work with multiple related processes. However, this method relies on the copy-on-write mechanism, which may significantly slow down the application when dealing with a large number of clients.[3] Due to these multiprocessing challenges, this application has opted for threads. To implement threads in the application, the POSIX Threads (Pthreads) standard is utilized.

d. Database interaction using SQLite

We will use a database to store the client's data (such as speed, location, username, and password) and a database to store the map more efficiently. Those will be updated and queried by using SQLite.

3. Application Structure

3.1 Communication Protocol

The communication protocol is implemented as simple as possible, the client and server communicate thru a concatenation of strings with different separators (in order to understand who sent the message in case of debugging). Each string has a starting word that describes the command, afterwards, if needed, the arguments for the command are provided using separators and in the end, the last word describes the SQL query the server will send to the specific table.

a. register

During registration, the following input will be collected sequentially: username, first name, surname, password and subscription preferences. For each attribute, the server will validate the input and collect it to send it to the database.

b. login

The server will perform a query to the database using the input provided by the client in order to check if there is a user registered with those credentials.

c. report

The input regarding the event will be broadcasted as a struct data type to each client. Each client will cache the event in a local array depending on its details.

d. get-events

The client will iterate its events array in order to print the information.

e. get-info

The client will receive the client's speed/alerts.

f. subscribe

The client will modify its subscriptions [add/remove].

g. get-data

The client will gather the data from its active subscriptions.

h. help

The client receives the list of available commands.

i. exit

The client will disconnect from the server and exit the application.

3.2 Client-Server Architecture

As indicated in the figure below, the architecture regarding the client will implement 3 threads:

1. Command Thread: For reading commands, sending them to the server, and printing the delivered output
2. Warnings Thread: Receives updates about the speed of the user by continuously sending a 'get-info' command after considering authorization.
3. Notification Thread: For receiving notifications from the server regarding events reported by other clients by continuously sending a 'get-events' command after considering authorization.

The server will implement one thread for each client to simultaneously process their commands. In each thread, the server will receive commands from the client, solve them (probably by querying the application's databases) and then send the output back to the client[4][5].

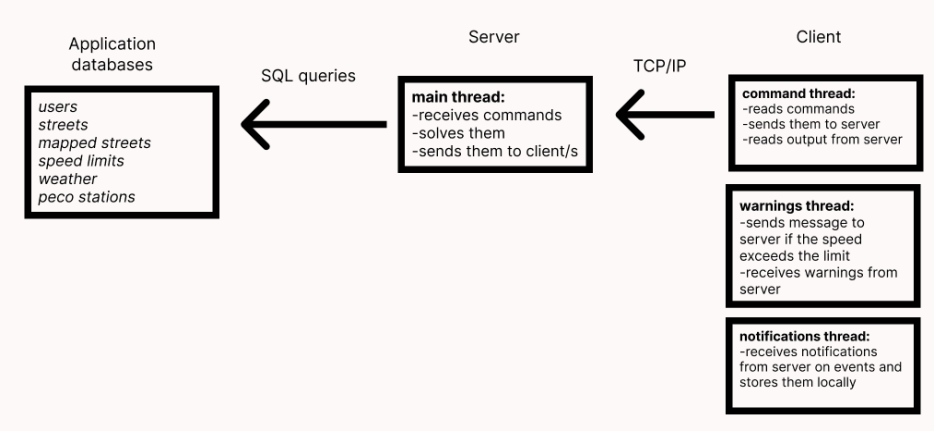

Figure 1: Client-Server Architecture

3.3 Database Contents

As the application's development has not reached the point of interpreting the command's data, the predicted design of the databases involved is:

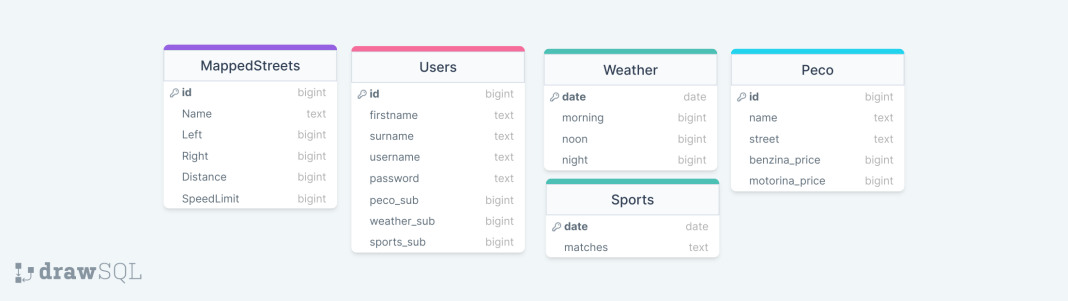

Figure 2: Predicted usage of databases[7]

4. Implementation Details

4.1 Speed update management

Each client will have a randomly assigned speed and street from the database initially. Afterwards, this application will update for each minute the location considering the speed and the map provided by the database. By using randomisation techniques and approximations, the server optimizes the number of updates, as it only makes them when it's time to send them to the client.

4.2 Sub-application for modifying and viewing the map within the database

I've constructed a small application that prints the database and can make modifications for the Streets table in order to be easy to track the modifications in speed within the "warning" thread and in order to ensure the data base describes the graph accordingly.

Figure 3: City map of Iasi and the intersections picked

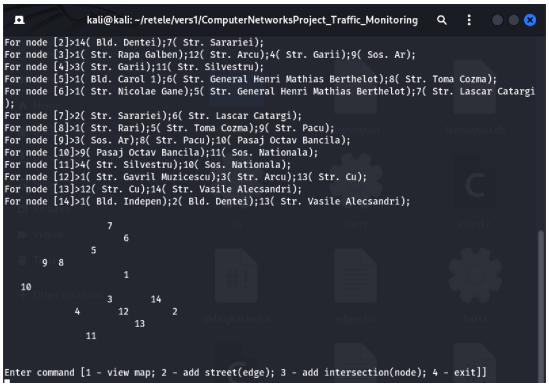

Figure 4: The visualization of the graph within the database

4.3 Data Interpretation

a. City Map

For the city map, we will use a database that we will interpret as a graph in which:

- Each intersection represents a node
- Each street between two intersections represents an edge
- Every edge has a length in meters and a speed limit

b. Traffic Events

For each traffic event, we will assign a random duration time in which it disturbs the traffic in that location for a more realistic approach. This duration time will be taken into consideration for additional warnings in case a client's location reaches an ongoing event.

5. Conclusions

5.1 Analysis of Potential Improvements of the Proposed Solution

From an efficiency point of view, I think it would be smart, although not necessarily optimal for the given small-scale project, to use another server that processes lower priority commands (such as subscribing to a radio station, weather information, etc.) to ensure the reliability of the operations of the main server with as many clients as possible.

Also from the point of view of efficiency, it is difficult to manage requests that involve broadcasting information to all clients. A system could be created by which for each incident the server would notify only nearby clients. Then clients that would not be interacting with that area would not be notified with unnecessary data. In a multi-client model, this change would improve the display time considerably, although it would use more memory.

I also think that an interesting improvement would be if police officers could make accounts in this application and send warnings/fines to other users or dismiss falsely reported traffic incidents.

References

[1] `https://linuxhint.com/posix-socket-with-c-programming/`

[2] Alboaie L. and Panu A. Computer Networks Course, `https://profs.info.uaic.ro/~computernetworks/files/6rc_ProgramareaInReteaII_En.pdf`

[3] UNIX Network Programming Volume 1, Third Edition: The Sockets Networking API By W. Richard Stevens, Bill Fenner, Andrew M. Rudoff, Published by Addison Wesley, Published Date: November 21, 2003, pg. 962

[4] Alboaie L. Concurrent Server Implementation, https://profs.info.uaic.ro/~computernetworks/files/NetEx/S12/ServerConcThread/servTcpConcTh2.c

[5] Alboaie L. Concurrent Server Implementation, https://profs.info.uaic.ro/~computernetworks/files/NetEx/S12/ServerConcThread/cliTcpNr.c

[6] https://patrickkarsh.medium.com/tcp-and-udp-a-comparison-7d43772b293e

[7] https://drawsql.app/