

# Monitorizarea traficului

Bogdan Zavadovschi

Facultatea de Informatică,  
Universitatea "Alexandru Ioan Cuza" Iași,  
bogdan.zavadovschi@info.uaic.ro

## Abstract

Monitorizarea traficului este un proiect care consta in implementarea unei aplicații prin care un participant la trafic poate transmite si primi informații despre diverse evenimente rutiere, transmițând totodata viteza și locația la un interval regulat de timp. Totodată acesta poate sa primească și diverse informații despre vreme, prețul combustibilului și ultimele evenimente sportive. Aceasta este alcătuită din două componente: un client și un server. Ambele folosesc tehnici de concurență, serverul acceptând mai multe conexiuni simultan, iar clientul poate transmite și primi date în același timp.

## 1 Introducere

Aplicația de monitorizare a traficului constă în două componente cheie: un client și un server, ambele implementate concurent. Clientul are capacitatea de a procesa mesajele venite de la server și de a transmite informații în mod simultan, fără a fi nevoie să se aștepte terminarea unei cereri. Totodată acesta **simulează** doi sensori: unul de viteză și unul de locație. Acești senzori vor fi interogați la un interval fix de timp pe care fiecare client îl va putea seta, în funcție de preferință, iar informațiile vor fi transmise către server. Pe baza informațiilor primite serverul va returna incidentele din trafic (dacă există în regiunea sa) și va modifica ultima locație a utilizatorului în baza de date.

Serverul are posibilitatea de a suporta mai mulți clienți simultan, prin implementarea concurență reușind să răspundă la mai mulți clienți în același timp. La un interval regulat de timp, acesta va trimite informații către toți clienții (care au selectată opțiunea) informații despre vreme, evenimente sportive și ultimele prețuri la combustibil. În cazul în care este raportat un incident, serverul va transmite tuturor utilizatorilor din regiunea specificată evenimentul.

Având imaginea de ansamblu, detaliile privind implementarea și arhitectura aplicației vor fi prezentate în următoarele secțiuni, totodată fiind prezentate și anumite îmbunătățiri pe partea de logică și performanță a aplicației.

## 2 Tehnologii utilizate

### 2.1 Protocolul de comunicare

Comunicarea dintre client/clienti și server este realizată prin **Transmission Control Protocol (TCP)**. Decizia de utilizare a acestui protocol provine din faptul că toate pachetele de informație trebuie să ajungă în aceeași ordine, ceea ce este important întrucât pe baza datelor primite de server este implementată logica de transmitere a evenimentelor rutiere, iar mesajele de tip text primite de client trebuie să fie inteligibile.

### 2.2 Stocare datelor

Pentru a putea ține o anumită evidență a datelor, atât a utilizatorilor, cât și a evenimentelor media, incidente rutiere, limite de viteză șamd. aplicația utilizează o bază de date, *sqlite3* ce asigură persistența și stocarea datelor. Folosindu-se de limbajul SQL, abstractizările la nivelul implementării sunt relativ ușor de făcut, scurtând timpul de implementare.

### 2.3 Securitate

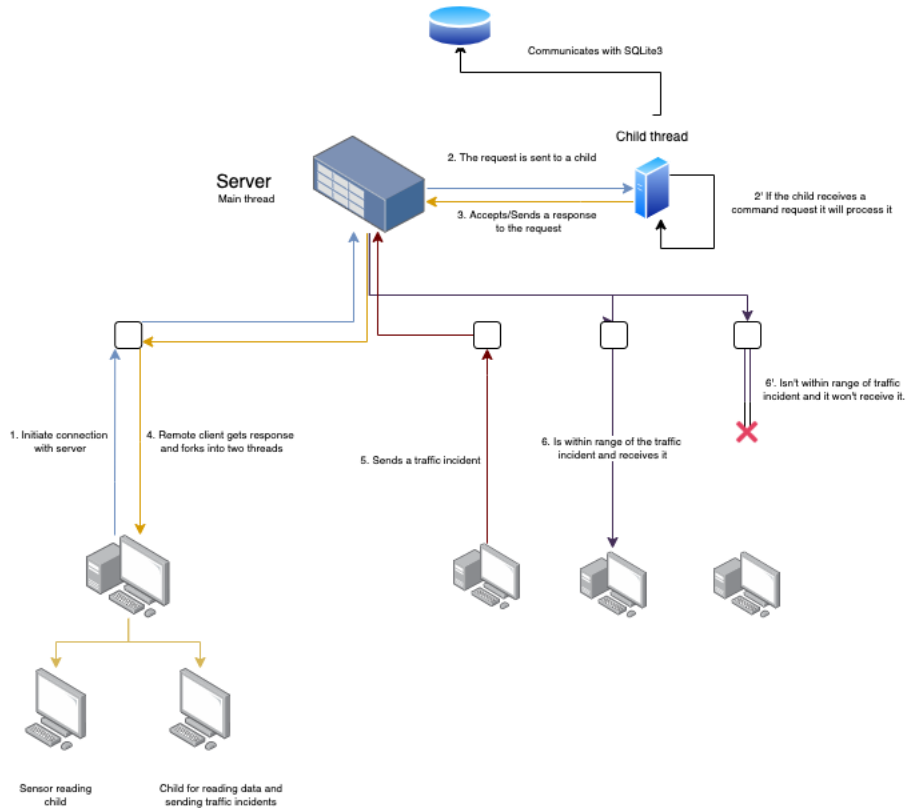
Din motive de securitate, odată ce avem conexiunea între client și server, vom crea un nou context de tipul SSL, transmitând astfel datele criptate. Pentru acesta vom folosi librăria *OpenSSL* din C++. Totodată stocarea parolele în plain-text nu este o soluție bună pentru zilele noastre, astfel că din aceeași librărie vom folosi algoritmul de hashing **SHA-256**.

### 2.4 Simularea Senzorilor

În încercarea de a face cât mai ușoară tranziția de la o aplicație de calculator la una de mobile, am adăugat un feature prin care se pot simula doi senzori, unul de viteză și unul de locație, strâns legați între ei. Aceștia pot fi citați, returnând date cu un anumit nivel de corectitudine între ele.

### 3 Arhitectura aplicației

#### 3.1 Diagrama aplicației



Pentru a putea explica diagrama de mai jos vom împărți în două secțiuni: clientul și serverul

#### 3.2 Arhitectura Clientului

După cum se poate observa încă din diagramă clientul este bazat pe o arhitectură concurentă, având următoarele caracteristici:

1. Inițial părintele încearcă să stabilească o conexiune cu serverul, folosind socketuri.
2. Dacă serverul acceptă conexiunea, în client se va forma un nou thread care va folosi socketul părinte să trimită informațiile despre viteză și locație, în timp ce părintele va putea să trimită și să primească informații rutiere sau media.
3. O dată la un interval de timp prestabilit se vor trimite informațiile despre viteză și locație

4. La un alt interval de timp, clientul va primi informații media (dacă a selectat această opțiune)
5. Dacă un client raportează un incident serverul îl va procesa și va încerca să facă broadcast la toți clienții aflați în aria în care s-a raportat incidentul. După cum se poate observa în pașii 6 și respectiv 6' informația nu este transmisă la toți clienții.

### 3.3 Arhitectura Serverului

Serverul folosește tot o arhitectură concurentă, prin care reușește să suporte atât mai multe conexiuni simultan, cât și să răspundă la mai mulți clienți în același timp.

1. Prin **multiplexarea I/O** se vor monitoriza descriptorii socketurilor asociate clienților remote, creându-se un nou thread la momentul în care o cerere nouă ajunge. Astfel putem suporta mai multe cereri simultan.
2. Primul pas prin care o cerere trece este procesul de autentificare, urmând ca mai apoi să fie validată comanda, iar în final să se încerce procesarea informației.
3. Pentru validarea autentificării folosim tokenuri generate random, care sunt stocate în baza de date și sunt atribuite unui utilizator. Pentru a putea evita orice fel de manipulare a datelor, tokenul este format din două componente: Identificatorul (care se află stocat în plain text în baza de date) și Validatorul (care se află într-o formă **hashed**, utilizatorul fiind singurul care are access la varianta plain text.)
4. Dacă autentificarea a avut success se va verifica dacă comanda dată de utilizator există, iar în caz afirmativ se va executa logica.
5. În final, indiferent de rezultatul execuției, serverul va trimite un răspuns clientului, urmând ca mai apoi să fie închis thread-ul ce s-a ocupat de răspuns.

## 4 Detalii de implementare

## 5 Concluzii

Aplicația **Monitorizarea Traficului** implementează un set de feature-uri destul de minimalist, existând loc de îmbunătățiri atât pe partea de securitate cât și pe cea de performanță a aplicației.

În primul rând, pentru a putea să scalăm aplicația ar trebui modificată baza de date din **sqlite3** într-o bază de date de tipul NoSQL (precum MongoDB), iar pentru stocarea datelor de autentificare (token-urile) ar trebui folosit un in-memory database precum Redis (întrucât necesită un număr relativ mare

de interogări, mai precis este strâns legat de numărul de cereri pe care le face utilizatorul).

În al doilea rând, generarea unor thread-uri pentru fiecare cerere poate deveni destul de exhaustivă, astfel că nu am putea scala prea mult aplicația fără a avea probleme majore. O implementare mai bună ar fi prin folosirea unor thread poll-uri, articol ce poate fi regăsit la punctul 3 din Bibliografie.

În final, pe partea de securitate a aplicației, am putea implementa JSON Web Tokens folosind diferite Key Set-uri, ceea ce ar ajuta foarte mult la scalarea aplicației, în cazul în care ne-am dori mai multe servere să fie pornite simultan.

## 6 Bibliografie

1. SSL Server-Client using OpenSSL
2. Algorithm for offsetting coordinates by some amount of meters
3. Using ThreadPools in C