- Javier Bastande Leon

- Student Id: 23176989:

- GitHub Repository: github.com/JavierBL89/Smart_Traffic_Management

- Programming Language: Node.js

- IDE: Visual Studio

Smart City Crossroad Traffic Management System

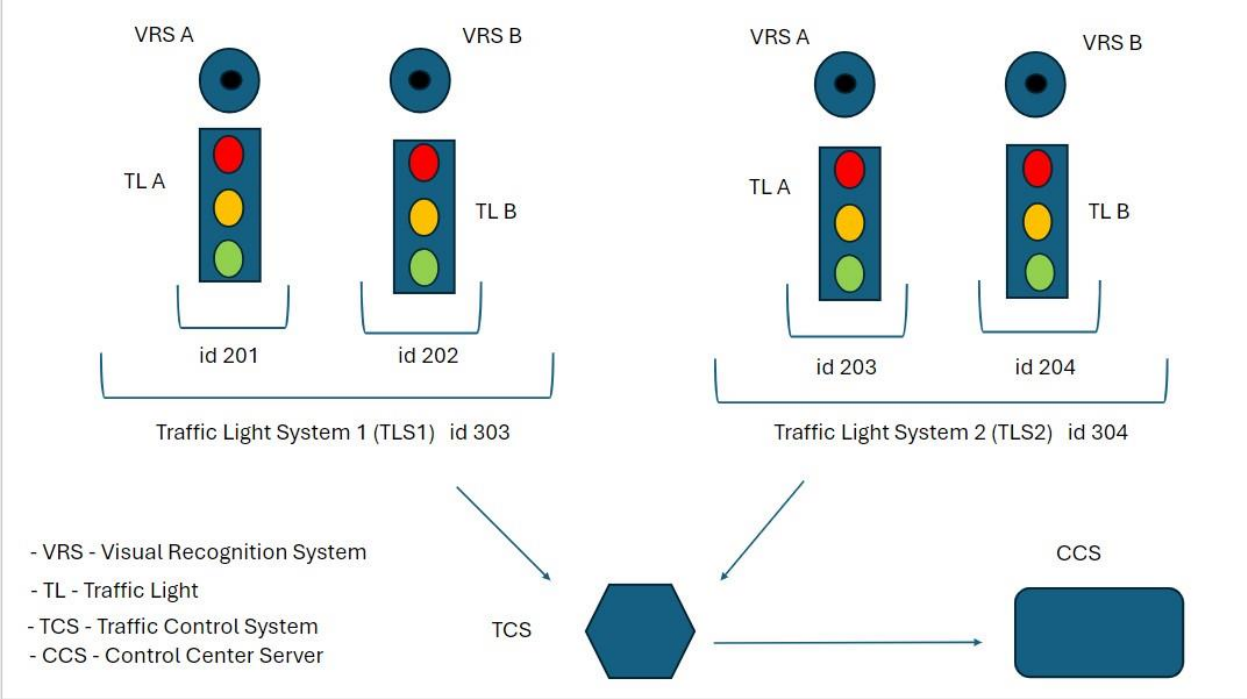
Table of Contents

Project Proposal

Smart Traffic Management System for a *crossroad* of a Smart City is to optimize the traffic flow and improve road safety.
The project is a conjunction of systems that collect real-time data to then be analyzed and perform operations based on that data to efficiently and smartly address different situations such as inefficient traffic light control timings in current cities which have standard time intervals in which they operate “green/red” without monitoring the traffic density and interrupting the traffic flow.

This Traffic Management System is composed of 3 main entities and their components:

- **Control Centre System**(Software layout*)*: Manages the integration, initialization, and operation of Traffic Control Systems.
- **Traffic Control System** *(Software layout)*: Manages the integration, initialization and operation of Traffic Light Systems *(Software layout)*. It orchestrates the behavior of the Traffic Light Systems based on traffic data analyzing.
- **Traffic Light System** *(Physical layout)*:
    - Traffic Light
    - Visual Recognition System



This Traffic Management System is equipped with very basic algorithms.
The system can be configured to run *n* number of traffic control cycles, for a desired time defined in seconds.
During each traffic control cycle, each Visual Recognition System (VRS) performs three traffic scans. The results from these scans are sent to the Traffic Control Server(TCS), which analyzes the total number of vehicles counted from each Traffic Light System (TLS).

Based on this data, the TCS determines the state (green or red) of each TLS for the upcoming cycle. A key feature of the algorithm is that the green phase can be extended for 5 seconds up until 3 times before moving to the next cycle. It prevents against maintaining the same state for any TLS for long periods of time. This mechanism ensures that if one TLS consistently reports higher traffic density, the system allows traffic to flow from the other TLS, which reports less traffic, thus balancing the traffic load.

When the Traffic Management Cycle is initially activated, the system starts with a predefined state: TLS 1 is set to "green" and TLS 2 to "red". In the first cycle, the VRS systems collect traffic data and reports it back to the TrafficControlManager, which then analyzes this data, determines the traffic density and passes this info to a smartCycle() which it then determines whether if the green phase should be extended or not, and initiates transition to the yellow phase after green phase ends. This process continues until the system reaches the maximum number of traffic control cycles configured by the user, ensuring the system does not run indefinitely.

System Components Overview & Specifications:

1. Visual Recognition System

This service utilizes visual recognition cameras to monitor traffic density, capturing the number of vehicles within a specified period of seconds.
The camera performs a traffic scan (vehicle counting) three times to complete a whole scan cycle before sending the data to the Traffic Control System. Continuous traffic monitoring helps with the assessment of real-time traffic density and makes it more accurate. In addition, future implementations of AI models can identify patterns at different times of the day, and days of the week or year.

1.1 Key features:

- Monitor real-time traffic density
- Visual recognition camera for vehicle counting
- Collects and report data periodically

1.2 Service functionalities:

- **Vehicles counting**: Through visual recognition the system counts the number of vehicles during a period of seconds.
- **Anomaly detention:** System identifies traffic anomalies. E.g.: If a vehicle is not in

motion for a period of time while the traffic light is “green”, or a vehicle if occupying both more than one lane might indicate an anomaly, and can report the anomaly for human monitoring or alert emergency services.
- **Data report**: System reports a detailed output of the number of vehicles and different types of vehicles recognized during a scan cycle.

2. Traffic Control System: The system integrates the core physical layout (traffic light systems) of the Traffic Management System that services interact with to manage traffic flow. For this specific scenario the Traffic Control System is composed by the integration of 2 different “Traffic Light Systems” which in turn are composed of 2 individual traffic lights that work in conjunction forming a “Traffic Light System”.

It acts as the central hub for the traffic lights state operations. It receives the instructions from the Control Center and manages each “Traffic Light System” at the intersection(crossroad)

2.1 Key features:

- Automated traffic light control based on real-time data received from Control Center
- Dynamic updates of state timings to optimize traffic flow.

2.2 Service functionalities:

- Receive commands from Control Center: It listens and receives instructions on traffic light states and timing settings from Control Center.

- **Analize traffic data**: Analyses received data and assesses current traffic density.

- **Coordinate Traffic Light Control**: Based on the data analized and decision-making operations, the service determines the most optimal traffic light state for the next cycle (“green, yellow, or red”).

- **Traffic anomalies detention**: Detects and reports traffic anomalies detected such as accidents or stopped cars when traffic should flow.
- **Report state update to control Server:** Provides real time feedback status which ensures synchronization of events between the TrafficControlManager and the sever.

3. Traffic Light System: System manages a set of traffic lights and cameras at a traffic junction. It helps control and monitor traffic effectively. It sets up the traffic lights and cameras, manages their operations, and gathers traffic data.

3.1 Key features:

- Controls traffic light states
- Report accurate traffic data

3. Control Center System: Serves as the central command of the Traffic Management System, managing Traffic Control System (TCS) initialization and configuration The Responsibilities:

3.1 Key features:

- Service Initializes Traffic Control System and its components.
- Configures traffic control cycle parameters
- Launches and manages traffic control cycles

Message Formats, Data Exchange & Communication Protocols

1- Traffic Control Initialization (init_traffic_control_system.proto)

Description:

Initializes all traffic control systems associated with the server. Synchronously initializes traffic control systems and returns a response.

```
// service definition
service InitTrafficControlSystem {

    rpc InitTrafficControlSystem (InitRequest) returns (InitResponse) {};    // response will include the payload in {}

}
```

Message definitions:

-Empty request:

-Response: message attached to the event repsonse: string

```
//Request
message InitRequest{
    // Empty
}

// Response
message InitResponse{

    string message = 1;
}
```

2- Traffic Control Configuration (confic_tcs.proto)

Description:

Configures parameters for traffic control cycles.

RPC: Bidirectional stream.

```
// Service definition
service ConfigTrafficControlSytem {

    rpc ConfigTrafficControlSytem (stream ConfigRequest) returns (stream ConfigResponse) {};
}
```

Message definitions:

-Request: configure green cycle length in seconds: integer.

-Request: configure total number of traffic control cycles: integer.

-Response: bolean representing the operation outcome: true/false

-Response: message with operation outcome: string.

```
// Numbers to be added
message ConfigRequest {

    // The number of traffic scans within a scan cycle
    int32 greenCycleLength = 1;

    // The length of each traffic scan in seconds
    int32 numbOfTotalCycles = 2;
}

// Response
message ConfigResponse {

    // Configuration update confirmation
    bool status = 1;

    // Confirmation message for each parameter update
    string message = 2;
}
```

3 – Traffic Report Configuration (traffic_report.proto)

Description:

Service allows user to enter a stream of desired parameters collected during the cycles of the traffic control systems to be reported after TLS operation ends. After user selection, server responds with a single message with confirmation of the report configuration.

RPC: Client Stream

```
// Service definition
service TrafficReport {
  rpc ConfigureReport (stream ReportConfig) returns (ReportResponse);
}
```

Message definitions:

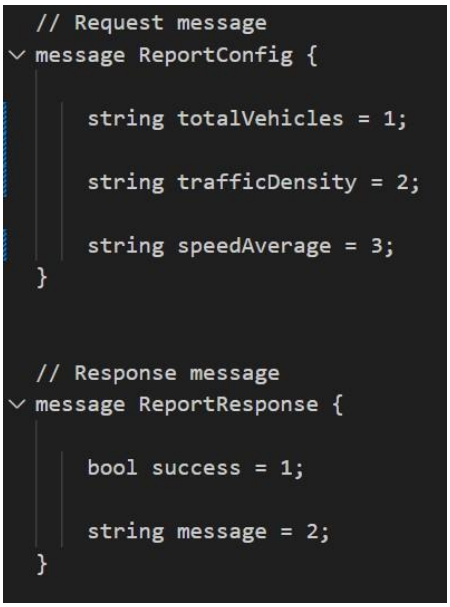
-Request: config if total vehicle must be added to the report: string

- Request: config if traffic density must be added to the report: string

- Request: config if speed average must be added to the report: string

- Response: bolean representing the operation outcome: true/false

- Response: message with operation outcome: string.



```
// Request message
message ReportConfig {

    string totalVehicles = 1;

    string trafficDensity = 2;

    string speedAverage = 3;
}


// Response message
message ReportResponse {

    bool success = 1;

    string message = 2;
}
```

4- Start Traffic Control Operations (start_traffic_control.proto)

Description:

Initializes smart traffic control operations.
User makes the request to initiate a traffic control cycle with a single input.
The sever listens to events during the cycle and reports a stream of messages in response to the client.

RPC: Server Stream.

```
// service definition
service StartTrafficControl {

    rpc StartTrafficControl (StartRequest) returns ( stream StartResponse) {};    // response will include the payload in {}

}
```

Message definitions:

Empty request:

On response for each event listened to

- Response: bolean representing the operation outcome: true/false
- Response: message attached to event: string.

```
// Numbers to be added
message StartRequest{
    // Empty
}

// Response
message StartResponse{

    // initialitation status
    bool status = 1;

    // Confirmation message
    string message = 2;
}
```

Communication Protocols in Application Interface

1. *Urinary* intit_traffic_control_system.proto

2- *Bidirectional Client-Server Streaming* protos/config_tcs.proto

2. *Client Stream* protos/traffic_report.proto

4. *Server Stream* protos/start_traffic_control.proto

When the application starts, the text-based UI is displayed

```
1- Initialize Traffic Control Systems (Urinary) (InitTrafficControlSystem
service)'
```

```
2- Configure Visual Recognition Systems 'Systems must be previously initialized'
(Bidirectional client stream - server stream) (ConfigureVisualRecognitionSystems
service)
```

```
3- Configure desired data shown on traffic reports' (Client Stream)
(TrafficReport service)
```

```
4- Start traffic control cycle' (Server Stream) (StartTrafficControl service)
```

Error handling

GRPC provides built-in mecanishms to handle errors efficiently, which are used in the systems to manage exceptions occurred during operations.

- Each call with gRPC returns a status code and a message, each code provides accurate information of what the problem is, a network failure, an unavailable service or any error related to data.

Client side

```
// handle any error during communication
call.on('error', (error) => {
    console.error('\nError from server:', error.message);

})
```

Sever side

```
// handle any error during communication
call.on('error', (error) => {
    callback(null, { status: false, message: `Error: ${error.message}` });
});
```

- **Try-catch blocks**:

All methosd on server side are wrapped in try-catch block to capture exceptions during execution of RPC calls. If an error occurs, the status and the message are sent back to the client.

```
StartTrafficControl: (call) => {
    let trafficControlManager;
    let trafficDataReportManager;
    try {

        // iterate through the list of traffic control systems associated to C
        for (let tcs of controlCenterInstance.getListTCSManager()) {
            trafficControlManager = tcs.getTrafficControlManager();    // grab
            trafficDataReportManager = tcs.getTrafficReportManager();  // grab

        }
        // start traffic control cycle
        controlCenterInstance.startTrafficControlCycle();
        handleTrafficControlCycle(trafficControlManager, call);

    } catch (error) { // error handling
        console.error('Error  while starting traffic control cycle:', error);
        call.write({ message: `Error: ${error.message}` });
        call.end();
    }
```

- **Stream error handling:** During stream procedures, both the server and the client listen for errors. This is handled by error events emited which logs the error details and finishes the stream smoothly.

- **Asynchronous operations and Promise error handling**: The systems widely uses javascript promisses for non-blocking operations.
Each asynchronous operation is wrapped in a try-catch block to effectively handle and capture any error occurred when a promise is rejected. Also, when an error occurs, it is propagated back to the caller through the rejection (*Error propagation*).

Advance Features

- GRPC supports **concurrent streams** which allows the system to manage multiple streams of data simultaneously.

- **Service registration** allows systems to be discovered and connect.

- **Protocol buffers** reduce the payload and serialize/ deserialize very fast improving overall performance.

- **Event architecture**. System is responsive to events and can react in real time to any changes detected.