WEST UNIVERSITY OF TIMIŞOARA
FACULTY OF MATHEMATICS AND COMPUTER SCIENCE
BACHELOR STUDY PROGRAM: Computer Science

BACHELOR THESIS

SUPERVISOR:
Lect. Todor Ivașcu

GRADUATE:
Mihai Andrei Gherghinescu

TIMIŞOARA
2023

WEST UNIVERSITY OF TIMIŞOARA
FACULTY OF MATHEMATICS AND COMPUTER SCIENCE
BACHELOR STUDY PROGRAM: Computer Science

Traffic Manager

SUPERVISOR:
Lect. Todor Ivașcu

GRADUATE:
Mihai Andrei Gherghinescu

TIMIŞOARA
2023

Abstract

The paper introduces an intelligent system for traffic signal applications that is designed to be as adaptive as possible and combines multiple technologies. At base, it is programmed on an micro controller and it uses fuzzy logic, but can be upgraded. This can be done by linking other hardware components to it with the help of a configuration file. So we can utilize multiple technologies like image recognition or speed/radio detection and get the best from all of them.

The system aims to be used in real time traffic scenarios, as an affordable and upgradable option, to decrease the waiting time at crossroads. This would lead to shorter travel times from point A to point B and would have a significant impact not only on our daily lifes, but on the environment as well.

Contents

1 Introduction **6**

2 Overview of the Technology Developed **7**
2.1 Object recognition based systems 9
2.2 Vehicle sensor detection based systems 10
2.3 Traffic Lights Synchronization based systems 11
2.4 Fuzzy Intelligent Traffic Signal Control System 12
2.5 Dedicated Short-Range Communications systems 13

3 A new flexible economic approach **15**
3.1 Proxy 17
3.2 Traffic Observer 19
3.3 Vehicle Tracker 20
3.4 Junction Main Server 21
3.5 Emergency states 25

4 Implementation details **26**
4.1 Common 27
4.2 IPC 28
4.3 CarDetector 32
4.4 ObjectDetectionServer 32
4.5 Proxy 34
4.6 JunctionMainServer 35
4.7 TrafficObserver 37
4.8 VehicleTracker 38

5 Further Directions **39**

List of Figures

2.1 Rasberry PI based system (Image Source ©) 8
2.2 Object Recognition Based Traffic System Model (Image Source ©) 9
2.3 Sensor Based Traffic System Model (Image Source ©) 10
2.4 Traffic Signal Synchronization Model (Image Source ©) 11
2.5 Dedicated Short-Range Communications Radio (Image Source ©) 13
2.6 DSRC Based System Model (Image Source ©) 14

3.1 All UC Diagram 16
3.2 UC: Connect 17
3.3 UC: Disconnect 18
3.4 UC: Find cars from provided images 19
3.5 Car tracking performance 20
3.6 System sketch 21
3.7 UC: Send vehicle data 22
3.8 Junction Phazes 23
3.9 Phaze switching example 24
3.10 Faulty phaze switching 25

4.1 Database Schema 27
4.2 TensorFlow 32
4.3 Proxy querys flow 34
4.4 Timer Increase/Decrease Impact 35
4.5 Faulty Scenarios Handling 36
4.6 Proxy junction data reply 38

Chapter 1

Introduction

Intersections are the points where two or more routes overlap with one another. As a result, for drivers, they act as an obstruction for smooth traffic flow especially within urban areas where the infrastructure demands them the most. This leads to unwanted traffic scenarios like interruptions, congestions, and overall poor control and management of the traffic.

To minimize the amount of time lost at crossroads while maintining the safety of the drivers, Intelligent Transportation Systems (ITS) were developed. One major impact that can be seen as a result is the reduction of CO_2 car emissions. You may think that this wouldn't be relevant when the era of combustion engines comes to an end, but there are so many other bennefits apart from this. For example, the less time it takes for resources to travel from a provider to a manufacturer, the faster the production can start so it also has a huge impact on the global economy.

As it's occurrence is noticed the most in urban areas, the problem is currently being tackled in concordance with the smart city concept. A smart city is defined as an ultra-modern urban area that aims to improve the overall life quality of citizens. It is based on different arhitectural approaches that involve modernizing and upgrading our current environment by applying new concepts and technologies on top of the ones currently in use.

Smart city traffic management refers to the implementation of advanced technologies and intelligent systems to optimize the flow of traffic within urban areas. It involves the use of various sensors, data analytics, and communication networks to monitor, control, and manage traffic conditions efficiently. The goal is to improve traffic flow, reduce congestion, enhance safety, and minimize environmental impacts. Smart city traffic management often includes the following key elements: ITS, Data Analystics, Adaptive Traffic Control Systems, Integrated Traffic Managemen, Connected Vehicles, Multi-modal Transportation, Smart Parking Systems, Sustainable Transportation

Chapter 2

Overview of the Technology Developed

To better understand the problem we must first know what fix attempts were used to minimize the negative effects of this issue through out the time and how they evolved.

The first fix attempt that was put in use was created long time ago and required the use of actual human resources to control the traffic. Policemans were responsible to manage the traffic at junctions. Once with the technological era, we were able to remove the need of this kind of resources by creating autonomous systems that alternate between STOP and GO phases. The design of the systems was pretty simple and straightforward, we would use lights to signal those phases. Red lights were matched with the STOP phase, green ones with the GO phase. To prevent collisions and maintain drivers safety amber was introduced as well. It notifys the drivers that STOP and GO phases will soon switch and they should take the corresponding measures.

As the time passed by and the number of cars on the road increased, drivers started experiencing traffic jams scenarios. Also there was clearly a better option then having a standard fixed time to change the lights. Many times the green signal was turned on even though there were no cars crossing that given road while on the other crossing roads there were actual drivers waiting. Something had to be done, a more efficient way had to be found, but at what cost? If we wanted to develop better solution we would need the help of expensive hardware components.

We reach the point in time when a new discovery was made: bugdet units based on microcontrollers, sensors and recievers [2] that were cable of running advanced algorithms (Figure 2.1). So all that was left was to develop new solution and use them to design an intelligent Traffic Light Control System (TSC)

By the time writing, TSC is an active research topic that proved to be

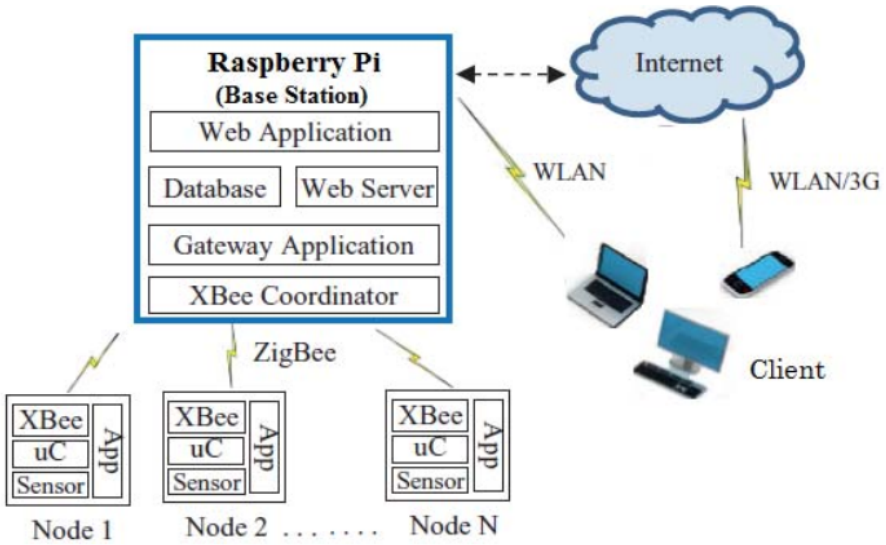


Figure 2.1: Rasberry PI based system (Image Source ©)

really challenging. Continuous work has been done on designing and developing intelligent traffic signal control systems that could address the issue.

So new methods and advanced systems and many other algorithms have been proposed by the researchers to solve this TSC problem, mostly beeing based on fuzzy logic, evolutionary algorithms, image processing, neural network, etc. [7]. All of the methods aimed to reduce the overall waiting time and prevent traffic jams from occurring while maintaining drivers safety. In this section we will discuss about some of the methods that were put in to use and their ups and downs. After describing each and every tehnology we will have a brief comparison to highlight some of their characteristics.

2.1 Object recognition based systems

One way to determine traffic scenarios was done with the help of cameras. To be more specific we wanted to calculate the traffic waiting queue by detecting cars present int the images provided by our hardware components (Figure 2.2). This problem was addressed by the help of vehicle tracking and image segmentation algorithms.

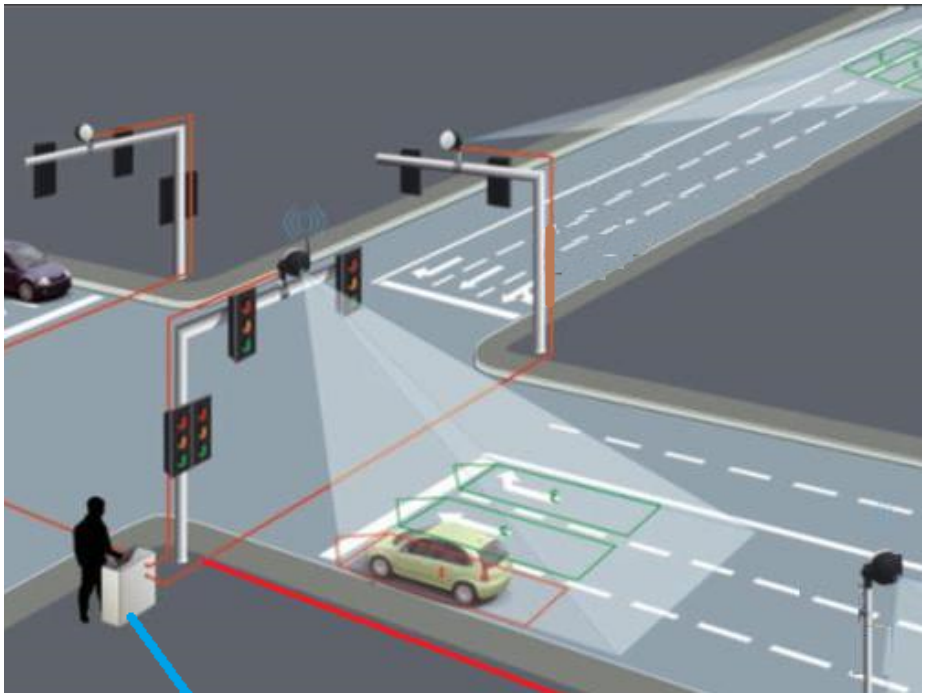


Figure 2.2: Object Recognition Based Traffic System Model (Image Source ©)

However, the techniques used to solve the problem proved to be ineffective in real-time scenarios due to it's high computational complexity that made the system unable to keep up with the traffic flow especially at high speeds. Also, some are not even able to operate in bad weather conditions, so they were doomed to failure. Studies had shown that whenever visibility is reduced due to rain, fog or other external factors that the pecentage of vehicles that will be detected will significantly drop. [5]

Even if the problems stated above were known, neural networks systems have been developed with the help of object recognition algorithms. They were designed to predict the upcoming traffic volume from a X-min daylight traffic flow on different traffic conditions. Like so we would be able to overcome the high computational complexity of the algorithm by avoiding recalculations. This lead to high memory needs, that could not be overcomed, so this approach

is unsuitable for real-time systems. Also, we believe that it is almost unpossible to predict traffic on a constant manner, and our best way to handle it is to dynamically adapt to it. We can still make use of a model that pools all traffic states for detection, but the actual "prediction" of the next traffic state should be calculated dinamically from real-time data.

2.2 Vehicle sensor detection based systems

Another approach would be to collect data about the vehicles approaching junctions with the help of GPS sensors. (Figure 2.3)

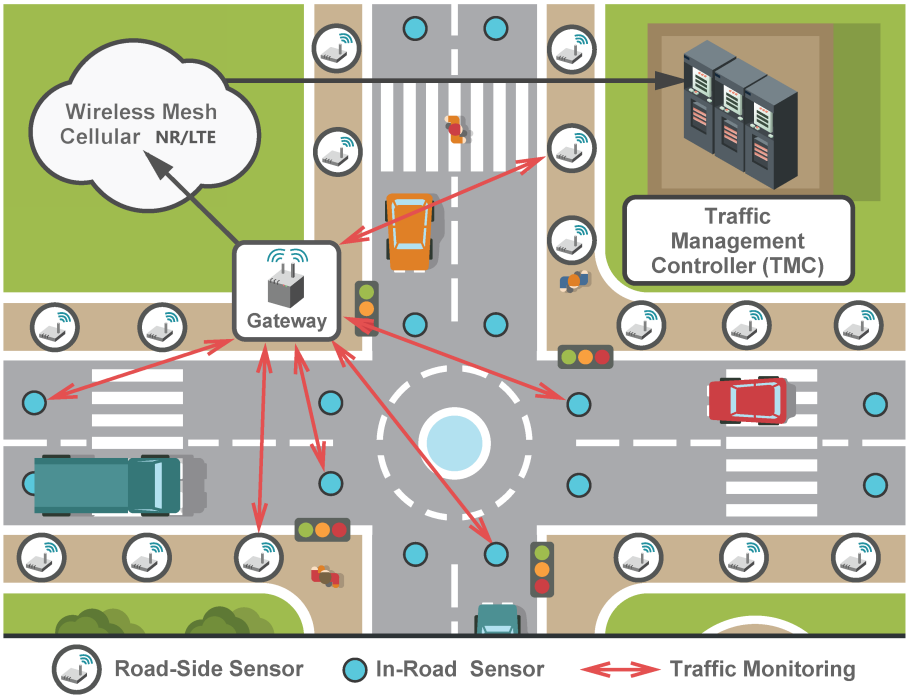


Figure 2.3: Sensor Based Traffic System Model (Image Source ©)

One way to do it is to track the position, speed and direction of the given vehicles. Sadly, this method would work only for a road network, it can not solely control the traffic signal timing for just one junction. This is due to the high speeds that vehicles travel at and the time complexity of the main algorithm.

Another way to do it is to monitor the arrival and departure of vehicles at a junction. This can be done with the help of sensors and traffic servers.

Like so we could use embedded technology to record the GPS data and send it to the traffic monitoring system through GSM/GPRS. The drawbacks of this method are the fact thats it involves very high implementation cost and, sadly, some vehicles can not be tracked using radio detection systems. This problem can be also aproached with the help of in-road sensors, but it would require even higher costs as you would need to often change the sensors because roads need to be rebuilt due to wear or ongoing costructions periodically.

2.3 Traffic Lights Synchronization based systems

Traffic Lights Synchronization (TSC) [7] systems aim to minimize the number of STOP and GO occurrences by adapting the traffic light phases at junctions. This technique involves all vehicle maintaining a consistent speed while traveling on one side of the road, allowing them to continue indefinitely to the opposite end of the road by consistently encountering green lights at intersections by maximizing their number. Studies had shown that in comparison with other fixed time and non-synchronized traffic control strategies it reduces overall travel time by up to 39%. [**ALEKO2019**]

Just like most of the others methods, it collects and procces data from real traffic scenarios to determine the green light timer. When vehicles pass through a juction the green light timer for the following juctions decreases by a certain amount (Figure 2.4). As with the other roads present at the juction, the red light time increase to guarante the continuous motion of already travelling cars. This can be done for mutiple levels of traffic depending on how many junctions do you want to synchronize with one another, but must of the the times it was handled as a 2 level system.

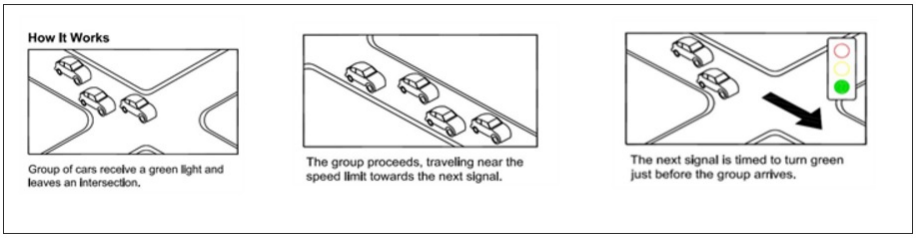


Figure 2.4: Traffic Signal Synchronization Model (Image Source ©)

Multiple approaches to implement this kind of systems were made with the use of varios AI algorithms and already known data collecting methods but the result are mostly the same. This method would lead to longer red signal time, but will reduce the overall waiting time when travelling long distances. One other advance to keep in mind is that, as result of this phenomenon, the wear of the vehicles will be drastically reduced.

The main disadvantages of this method is the fact that you will eventually have to prioritize one route. This would be problematic if multiple main

roads collide with one another. Also, the algorithm will not always provide an optimal solution as drivers following sideroads may have longer waiting times. One more thing to keep in mind is that this method is reliant on drivers speed consistency. Most of the cases, there will be a decent amount of speeding vehicles that will not be able to pass multile junctions without any kind of stops as the algorithm itself does not intend to handle those edge scenarios.

2.4 Fuzzy Intelligent Traffic Signal Control System

Fuzzy Intelligent Traffic Signal Control (FITS) [6] [3] systems were originally designed to mimic a human policeman in controlling traffic lights at an intersection. They are systems that take different types of traffic input and apply fuzzy rules to manage the traffic. Like so data is converted into fuzzy truth values between 0 and 1. For instance, the green light extension time can be modeled by a number of fuzzy sets including "none"(1), "short"(2), "moderate"(3) and "long"(4). The only thing that you are left to do is to define your membership functions for the given set. For example we can use the following functions to do so:

$$"none" - f(q) = max(min((10 - q)/10, 2), 0) \tag{2.1}$$

$$"short" - f(q) = max(min(q/10, (20 - q)/10), 0) \tag{2.2}$$

$$"medium" - f(q) = max(min(q/5, (30 - q)/5), 0) \tag{2.3}$$

$$"long" - f(q) = max(min((50 - q)/10, q/2), 0) \tag{2.4}$$

Like so we would alternate base on other fuzzy values or randomly, between choosing "none", "short", "moderate" and "long" fuzzy values, to determine the time that our green light should last.

Depending on the actual improvements of the traffic flow the algorithm will adapt, altering it's member functions and decreasing/increasing the odds to chose one specific fuzzy value from the set. One big advantage that this would lead to is that FITS mitigates the negative effects of detection malfunction by predicting the traffic states at the whole intersection by simulating real time traffic scenarios. So FITS may be able to run withot any kind of additional hardware, leading to huge cost advantages.

Despite all of the bennefits presented above, as with any AI based algorithm, it takes a lot of time for the algorithm to improve itself and the upgrade. Also the chance to upgrade is very reliant on traffic conditions. Furthermore, even if the algorithm has been running for a long time, it still can choose a non optimal solution that may or may not lead to traffic jams.

2.5 Dedicated Short-Range Communications systems

To explain why this kind of systems would work, first we have to understand what are Dedicated Short-Range Communications (DSRC) [8] [7]. DSRC are one-way or two-way wireless communication channels specially designed for use in automobiles that are mostly used by ITS to comunicate with other vehicles or infrastructure technology. They operate on the 5.9 GHz band of the radio frequency spectrum and are effective over short to medium distances.

One technique that was developed with the help of DSRC is the Virtual Traffic Light (VTL) approach. VTL is a biologically-inspired approach to traffic control that relies on Vehicle-to-Vehicle (V2V) communication by using the Signal Phase and Timing (SPaT) message and Basic Safety Message (BSM) from the DSRC OBU. (Figure 2.5)

Figure 2.5: Dedicated Short-Range Communications Radio (Image Source ©)

The radio is capable of broadcasting several messages defined by the SAE 2735 [1] protocol with the most important beeing BSMs. This kind of messages contain vehicle's current information (GPS location, speed and heading) that are associated with a temporary ID. As a result, by broadcasting BSM messages we are able to detect the vehicles approaching the intersection in a continuous

manner, unlike traditional methods that only detect the presence of the vechile using loop detectors.

You can image the cars beeing "moving routers", and the junctions beeing "stationary routers". Whenever one "gateway" or in our case one traffic route is overfilled with vehicles it blocks the others and starts letting them pass just like RIP protocol. Like so we are not actually introducing new sorts of technologys that may or may not fail, we just adapt already existing algorithms, that proved to be great solutions, to fit one specific use case.

Figure 2.6: DSRC Based System Model (Image Source ©)

With the help of this approach we can drastically reduce the implementation costs as it does not require any kind of additional expensive hardware components. Furthermore, it is robust against weather conditions and easy to mantain. Extensive simulations have shown that this kind of technology can reduce daily commute time in urban areas by more than 30%. Different aspects of VTL technology, including system simulation, carbon emission, algorithm design and deployment policy have been researched in the last few years. [4]

The main drawback of this approach is that it requires vehicles to support DSRC technologies, which might me problematic at the given time. Moreover, within V2V communications in VTL systems, there is a possibility of encountering partial obstruction by physical object present in the innermost Fresnel zone (non-LoS conditions). This presents a significant challenge in terms of making quick decisions effectively. However, even if DSRC technology isn't totally supported by vehicles and we might colide with non-Los conditions, it proved to be effective in reducing the average waiting time at juctions. So, maybe the next step on improving traffic flow, would be to create an infrastructure for this kind of technologie, but only time will tell. In the mean time the best thing we can do is to develop an economically efficient plan for transitioning from existing traffic control systems to VTL systems.

Chapter 3

A new flexible economic approach

We belive that the future of traffic light management will be based on DSRC like signals. No matter if it will still be based on DSRC, 5G or even 6G will take lead, we aim to provide a some kind of software system that would be able to handle any kind message based tehnology. Also, because at the given moment, the infrastructure for DSRC systems hasn't been fully developed and deployed we want to provide a economic approach to the given problem that would bennefit on the already working systems, but also can simulate the already working traffic light management protocol.

Even if DSRC will be fully deployed on a globas bases, there will still be alot of poor countries that will have a hard time upgrading their infrastructure. Because of that, we wanted to be able to have a working system even without investing any kind of money. Think about a PC for example, even one of the worst ones can run an OS and if you want to updgrade them, to make it run faster or handle a newer software, you would just add or replace a component with a better performing one. Not only you will be able to add "performance boosts" as needed to your given infrastructure but think about the scenarios when something goes bad? What if one of your component goes down and for example the traffic light will be stuck on red until someone comes and fixes it. That would lead to a HUGE traffic jam.

We also want to be able to take advantage of already working DSRC compatible vechiles without adding any kind of new hardware and be able to disable the additional protocols and move to a fully DRSC based system if needed.

Sounds great, right? This is how we designed it. The system will have **4 major components**: **2 types of clients**: one stationary one and one moving one, "attached" to our vechiles; **2 types of servers**: one that would manage the position of the clients and redirect them to the corresponding servers and one that would handle all of the messages and change the actual traffic lights.

For the simplicity on things we will name them as follows:

- **Traffic Observer (TO)** - the stationary client.
- **Vehicle Tracker (VT)** - the moving client installed on our vehicles
- **Junction Main Server (JMS)** - the server that handles the traffic lights
- **Proxy** - the server that will guide the clients to the corresponding servers

The system would work as follows: We would constanly have a TCP-IP connection between our clients and the corresponding JMS. Once they are connected clients would send various data about the traffic that would be further proccesed by the JMS and traffic lights would be changed accordingly. The TO would be responsible for sending data about the traffic on it's road and the VT would be responsible for sending it's own data to the server (Figure 3.1). We also handled the cases when there is an emergency ongoing like any kind of special vehicles crossing the junction but we will talk later about this because it requires some more attention as it can be easily exploited if not handled right. Just imagine any kind of cars beeing treated like an ambulance on duty, you would just be able to pass freely any kind of junction without ever waiting at the red light.

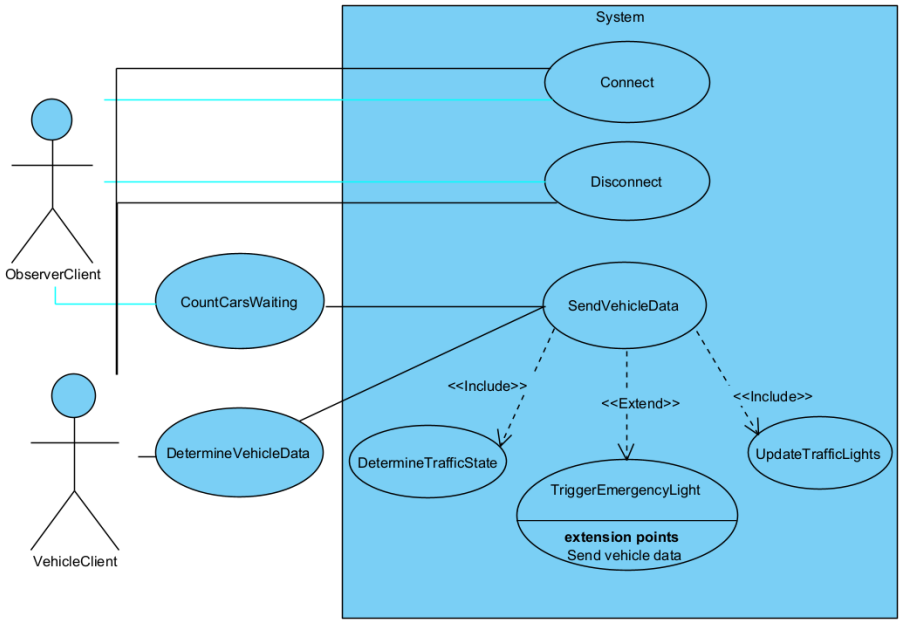


Figure 3.1: All UC Diagram

3.1 Proxy

The first problem that needs to be addressed is the fact that VTs will constanly have to switch between JMSs, but how do we know what server to connect to? We do not want to connect to each and every server in the covered area and start broadcasting mostly useless message. The answear would be that each client will know the main server off the system, that would act just like a proxy and redirect them to the corresponding JMS. For that we would need to store the JMS coordinates in one or multile database. The VT would need to just send its own coordinates and the direction they are heading, as a result the proxy would provide de IP address of the corresponding JMS (Figure 3.2).

As we want this to be scaled on a global level, the system supports horizontal scalling, as we can just stack proxys one on top of the other and instead of having stored only the JMS coordinates we will also store all the proxys that are in the subarea of the owner. To not overload any kind of servers we also implemented a load balancing mechanism that prevents this kind of issue.

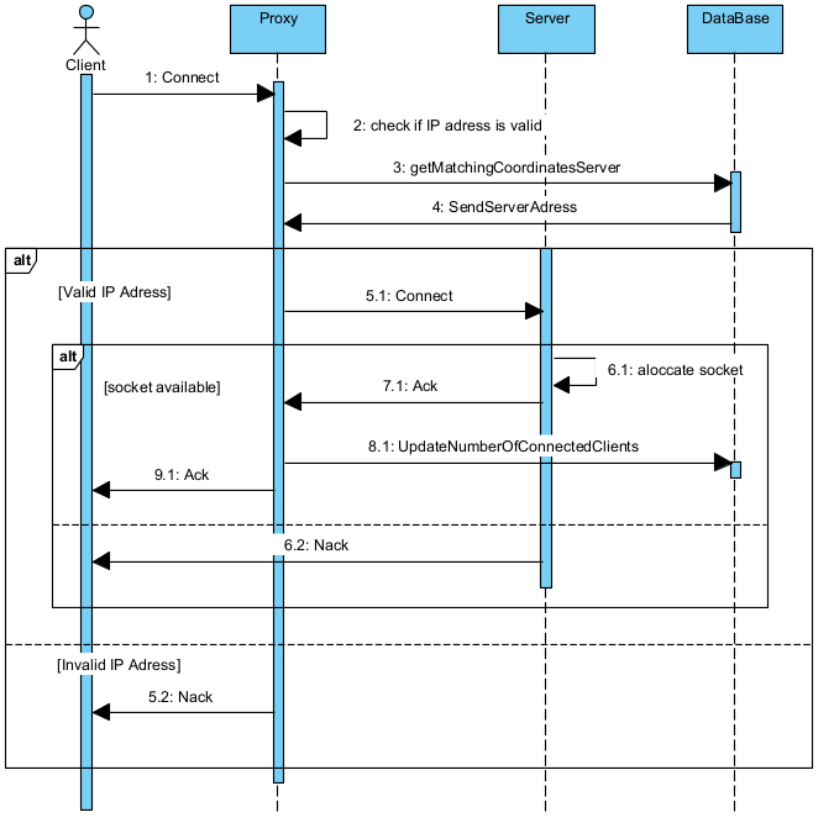


Figure 3.2: UC: Connect

One thing this can lead to is a better surveillance of the vechiles that aren't suppose to be on the road in the first place. The IP address of a given vechile can be "banned" and if it tries to connect to the JMS the message will be ignored and not only the JMS will not count the waiting vechile, sometimes leading to longer waiting times for the individual that is driving the vechile and broke the law, but also like this we would have the proof that he broke the law so we would be able to further punish him. We can lower the number of cars driving on the public roads that do not respect safety norms and reduce the number of accidents. For this to work we will need to also disallow owners to change their hardware without any kind of approval as a new radio would lead to a different serial number and this would be problematic.

Once passing the area of coverage of that given junction we would also need to disconnect from that given JMS (Figure 3.3). By doing this we will be able to always track the vehicles and react accordingly when one or multile leave the junction. One thing that we have to mention is that if we further think about this we could in theory keep a constant track of ALL the vehicles ON A GLOBAL BASES and no longer require licence plates, if, of course all the vehicles would support sending DSRC signals.

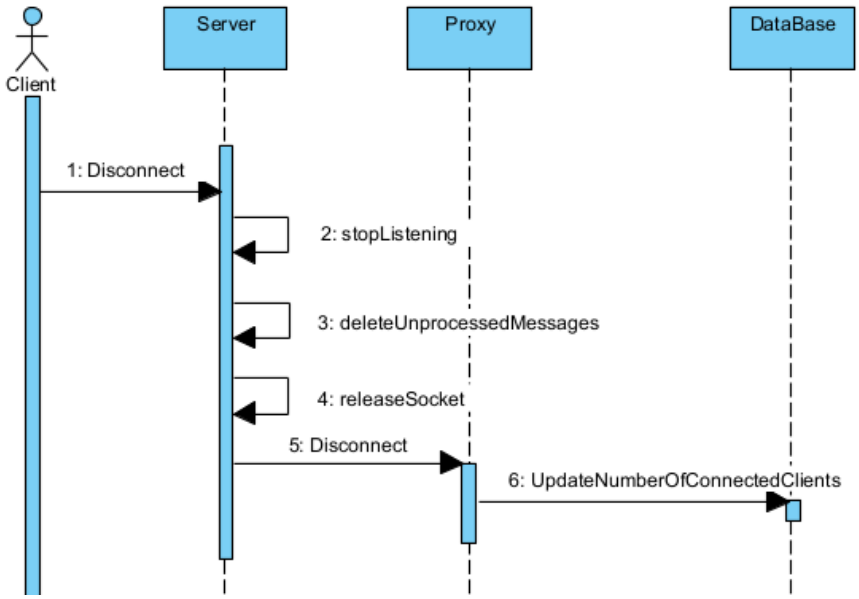


Figure 3.3: UC: Disconnect

3.2 Traffic Observer

The second thing we need to do is provide a way to detect the waiting vechiles withot the help of DSRC messages. This can be done in mutiple ways, but for the simplicity of things and to keep costs as low as possible we went with an camera based system. The way it works is as follows: multiple cameras are mounted on the road that are connected to microcontrollers, we take the images provided and apply object recognition algorithms to detect the passing cars.

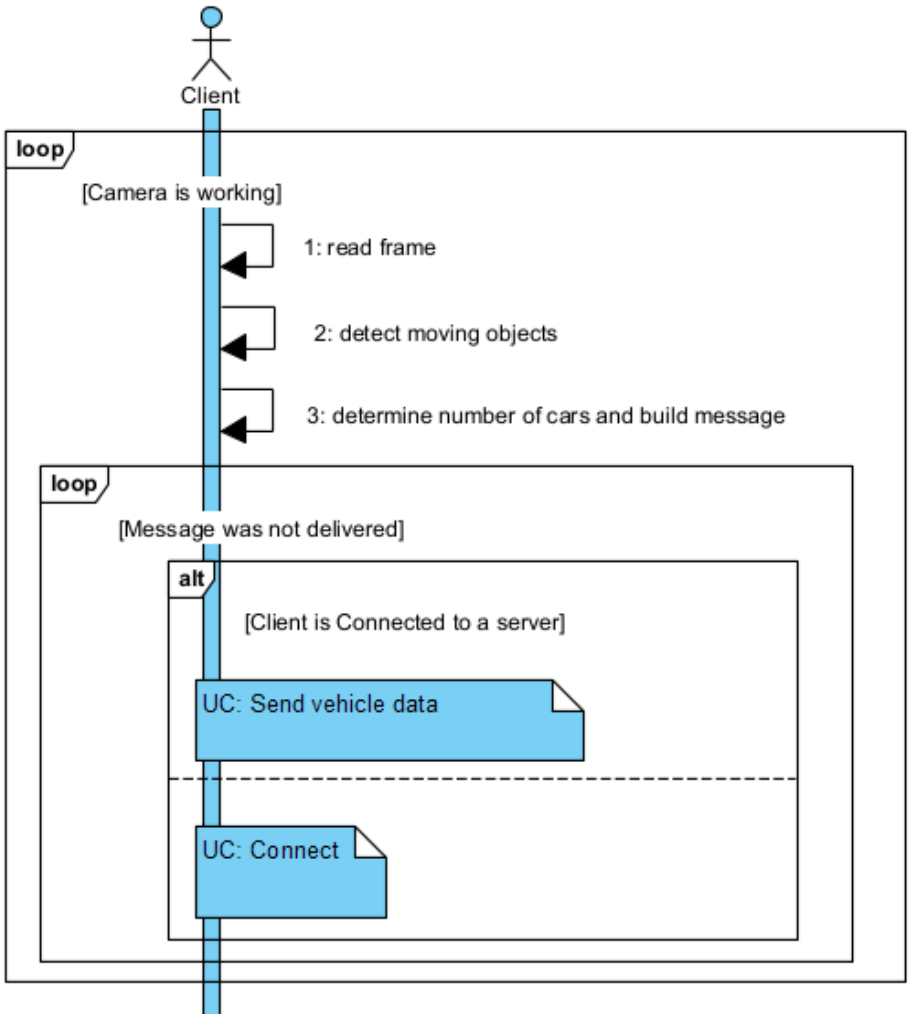


Figure 3.4: UC: Find cars from provided images

The first thing we actually do is preprocess the given images and detect the moving objects inside our frames and second of all determine if that moving object is or is not a vechile (Figure 3.4). Like this we can speed up the image detection process as we will have less pixels to keep track off. This overall speeds up the system as can be seen in Figure 3.2. The blue line represents the amount of cars detected with it running and the red one without it.

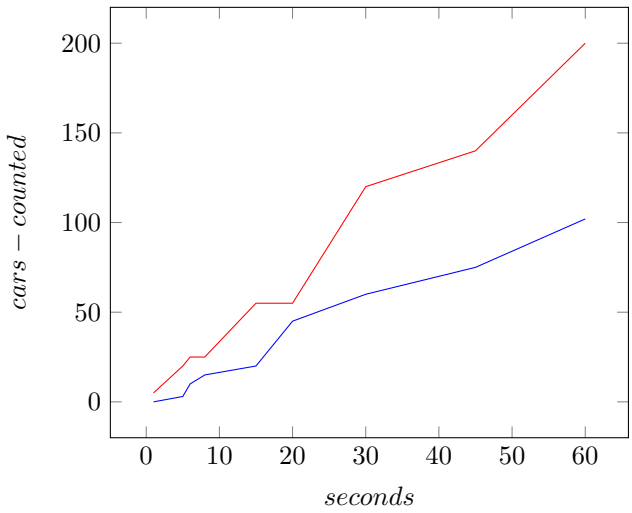


Figure 3.5: Car tracking performance

3.3 Vehicle Tracker

We want to be able to be able to send DSRC signals from any given vehicle. To do that, we make use of already integrated hardware components, to be more specific we make use of the radios and GPS installed on our vehicles. Like this, we get the coordinates of the vehicle as well as it's heading and the radio allows us to establish TCP-IP connection and send messages. So the only things that is left to do is to download our service and keep it running while travelling.

Whenever a vehicle wants to connect to a junction, it will have to specify it's serial number, like this we will be able to deny/accept it's connection. The only tricky part that is left to do is prevent anyone from changing thir serial number. This can be done in many ways, like changing the ACL of the executable to be ran or overwritten only by the system or simply putting a flag in memory to prevent any kind of acces in the location where the executable itself is stored or just encrypt the whole storage space of the vehicle. The security part of the system itself is not yet defined, it will be developed only if the project will gain momentum.

3.4 Junction Main Server

To handle all of those messages and change the actual flow of traffic we need to have a main command unit that acts as a server. Now that we know about all the components of the system we can start vizualing it(Figure 3.6). Clients will query the 2 time of servers and connect their corresponding junction. Once connected, they will send varios type of data to the junction server and the server itselft will change the traffic phases based on the data recieved.

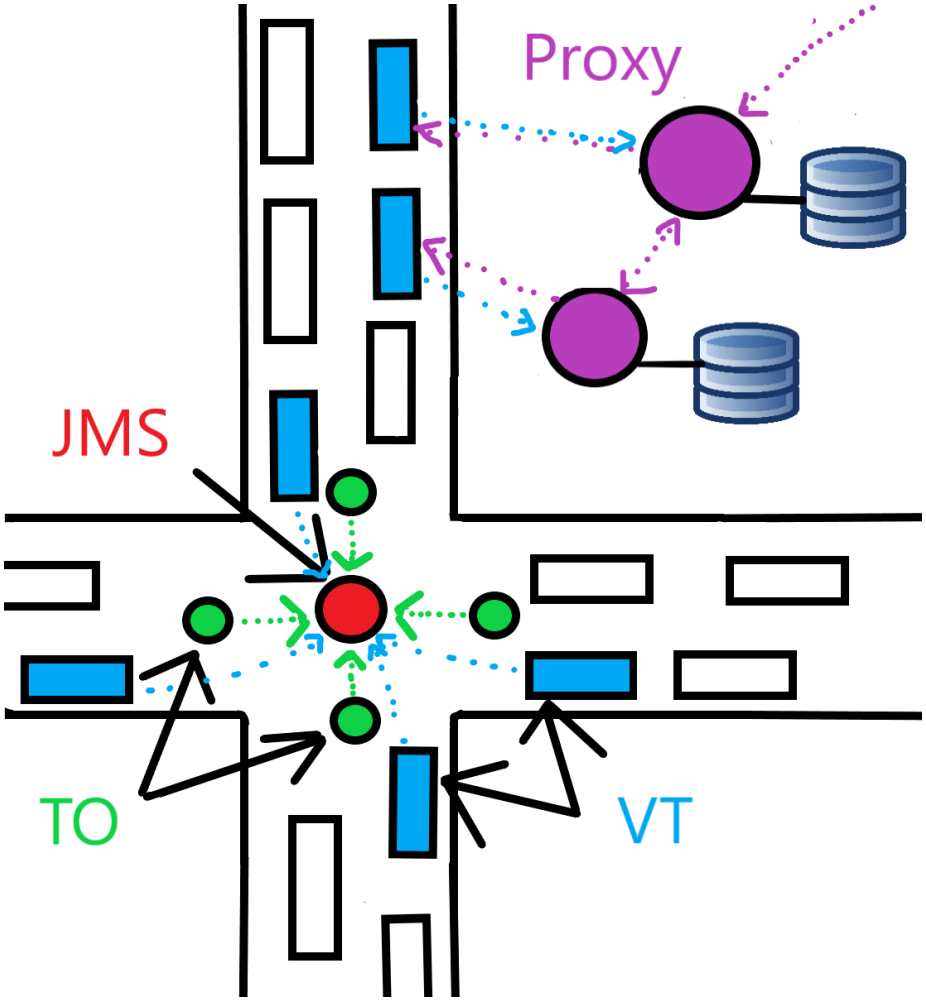


Figure 3.6: System sketch

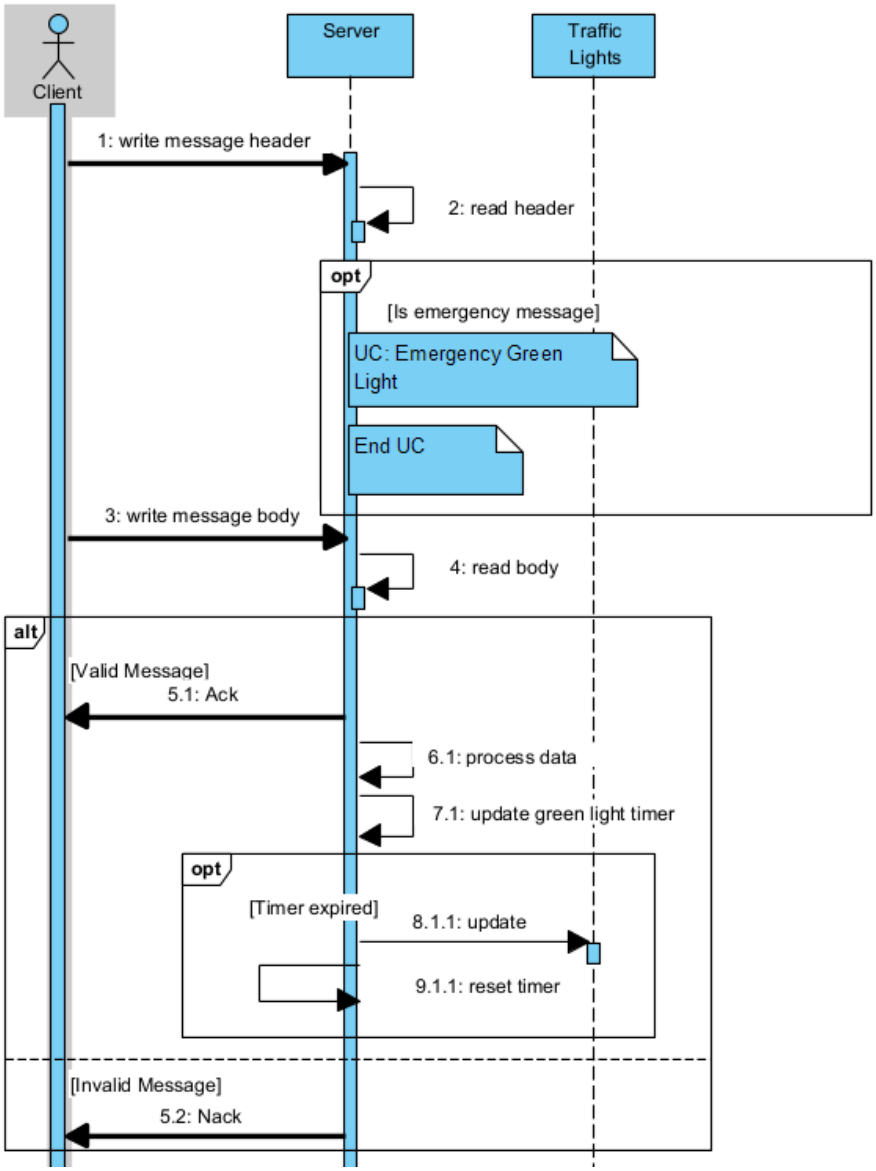


Figure 3.7: UC: Send vehicle data

The mechanism works mostly like any other TCP-IP mechanisms. We read the header of the message to determine the type of message recieved and the size of the actual message then based on the info we procces the data and update the traffic state. You may ask what happens if a car is detected by the TO as well as it's data is sent to the JMS trough a VT. Well, because of this scenario we chose to aproximate the traffic based on the number of cars determined by the

TO and the number of VT that sent messages to the JMS. Like this we make sure that, if one of the 2 components or both fail the system will still work. Also because noone is suppose to wait to long for the green light we can set a maximum timer for the red light. Imagine beeing alone on one road waiting for the green light and having thousands of cars crossing the intersection from another road, you would have to wait ages for you to be able to cross. This prevents that and acts just like an "aging mechanism" (Figure 3.7).

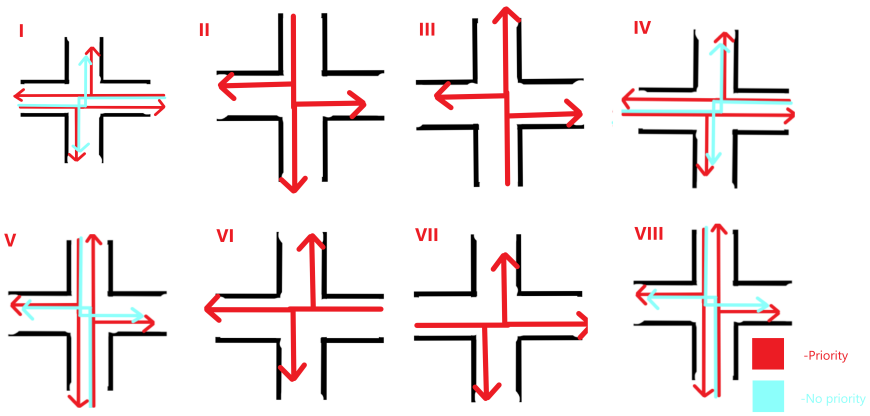


Figure 3.8: Junction Phazes

As far as the main algorithm for the traffic management goes we have 8 phazes of the traffic (Figure 3.8). The most basic scenarios is that the traffic will follow that 8 phaezes going trough phaezes 1 to phaze 8 in order and cycle. That's the implementation for the way the traffic works, but we want to also be able to jump from one phaze to any other at any given time, to be able to shorcut the system. For that we are basing on several things.

First and foremost we need to take into account the number of cars waiting. We do that by combining the provided data from the VT and TO and calculate an average of cars waiting.

Second of all we will need to fully understand our traffic pahzes, why the are required and how to determine the phaze we want to switch to. For that we first described the phazes based on the acting vehicles.

- PHAZE I: E + W waiting vehicles
- PHAZE II: N waiting vehicles
- PHAZE III: S waiting vehicles
- PHAZE IV: E + W waiting vehicles (same with PHAZE I)
- PHAZE V: N + S waiting vehicles

- PHAZE VI: E waiting vehicles=
- PHAZE VII: W waiting vehicles
- PHAZE VIII: N + S waiting vehicles (sane with PHAZE V)

We want to know when to switch from one phaze to another. For that we will have timers for each direction: E, W, N and S. At first all the timers will be set to a costum amount, but the waiting time will be changed dinamically based on the traffic conditions as well as the green light waiting time.

To start the whole mechanism we want to initiate the traffic as usual, having the normal order of phazez running and for us to switch to and abnormal phaze we'll just need one or more timers to expire. The check for the jump to an abnormal phaze will be done ONLY when GREEN LIGHT ENDS.

Last but not least we need to define the way our algorithm jumps from one phaze to another, for that we will use the following example (Figure 3.9):

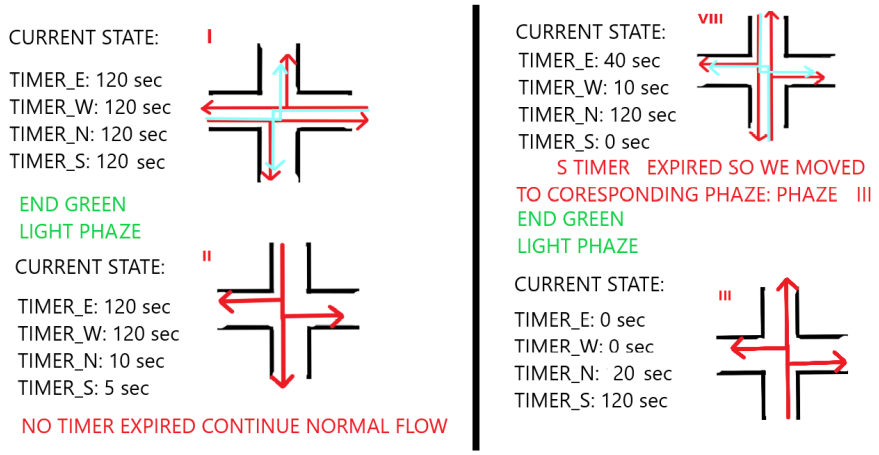


Figure 3.9: Phaze switching example

At the end of every green light phaze we check the timers. If any of them expired we jump to it's corresponding phaze. If, for example the following state coresponds to the phaze we want to jump to, we do not short circuit the normal flow of things. Every time we are trough green light phazes the timers corresponding to each lane that has a green signal is frozen and afterwards reseted.

There are also "traffic conflicts" that can occur, for example, what if both the timers from E and S lanes has expired, we can not give a green light to both lanes, there isn't a corresponding traffic state. This are the edge scenarios that we want to avoid the most. The way we handle them at the given moment is that we just use FIFO logic, take the timers that expired first and build an existing traphic phase. When this happens we afterwards update the green

and red light timers duration accordingly to prevent any more occurances. An example of the given scenario can be seen in Figure 3.10.

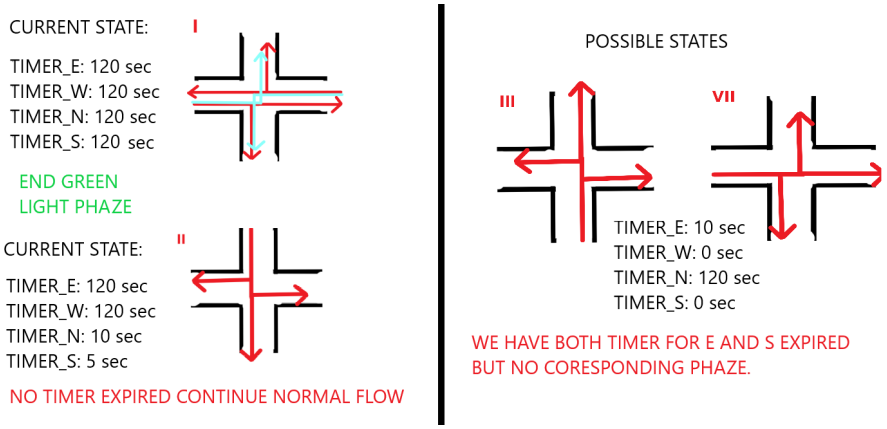


Figure 3.10: Faulty phaze switching

3.5 Emergency states

The system also recognizes and treats accordingly special vechiles like ambulances, police cars and fire trucks crossing the junctions. Whenever one of those vehicle is detected our server turns on the green light corresponding to the lane they are following and keeps it on until it crosses the junction. This is called and emergency green light state. If we are already in one to begin with, we will queue the next state by using FIFO rule. The whole communication is done only by using VTs as the process of detecting special vehicles in mission would take to much time when it comes to object detection algorithms and time is key for us in those critical moments. When no more emergency vehicles are present, tarffic will just go back to it's original flow, from where it left off.

Chapter 4

Implementation details

The whole system was developed using C++17, Python, Boost, GLFW, MSVC WinAPI, OpenCV, Tensorflow and MySQL. The system itself is treated as a big project and splitted into multiple submodules:

- static librarys
  - Common
  - CarDetector
  - IPC
- executables
  - servers
    - Proxy
    - JunctionMainServer
    - ObjectDetectionServer
  - clients
    - VehicleTracker
    - TrafficObserver
  - testing

This section aims to explain in detail what each and every one of this submodules were meant to do, how they were implemented some examples of using them and the main features that they provide. The project is not cross platform, it's Windows oriented at the moment because during the development of the whole system Microsoft Visual Studio 2022 MSVC was used as it provided a really quick and easy way of debugging my applications, but the code itself can be later converted to be cross platform using CMake, appart from the testing module, as it is using WinAPI to spawn and handle processes.

4.1 Common

The main ideea of the common submodule is to act as a helper library. It provides solutions to the following commonly found problems:

Multi threaded concurency: Thread safe structures:
- ThreadSafePriorityQueue
- ThreadSafeQueue

Handling GPS output: NMEA coordinates data parsing:
- GPGLL, lat, N/S, lon, E/W, time, A/V, A/D/E/M/N * checksum
- GPGGA, time, , N/S, lon, E/W, Position Fix Indicator, Satellites used, HDOP, MSL Altitude, Units, Geoid Separation, Units, Age of diff. corr., Diff. ref. station ID * checksum
- GPRMC, time, A/V, lat, N/S, lon, E/W, Speed over ground, Course over ground, Date, Magnetic Variation, A/D/E * checksum

MySQL interactions: C++ class tabel encapsulation and query handling

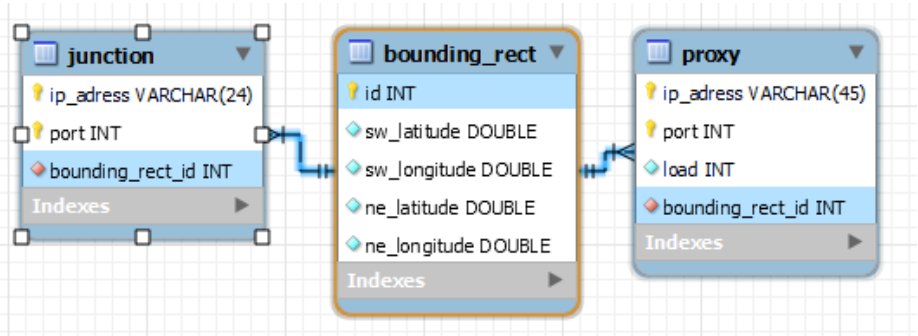


Figure 4.1: Database Schema

Parsing config files: Specific json parser, data type converter

Creating preplanned timed actions: Observers, Timers

Handling user input command line parser, signals handler

Syncronosly logging on multiple threads: Costumizable logger

4.2 IPC

To establish the communication amoung all the executables an costum made IPC system was develop, while using Boost Asio for socket handling. The loggic itself runs on 4 threads: one for keeping the context running, one for reading, one for writing and one for notifying progress.

```
template<typename T>
class Client
{
private:
    common::ThreadSafePriorityQueue<OwnedMessage<T>> incomingMessages_;

protected:
    boost::asio::io_context context_;
    std::thread threadContext_;
    std::mutex mutexUpdate_;
    std::condition_variable condVarUpdate_;
    std::unique_ptr<Connection<T>> connection_;
    std::atomic<bool> shuttingDown_ = false;
    boost::asio::io_context::work idleWork_;
    LOGGER("CLIENT");
public:
    bool connect(const utile::IP_ADRESS& host, const ipc::utile::PORT port);
    void disconnect();
    bool isConnected();
    void send(const Message<T>& msg);
    bool waitForAnswear(uint32_t timeout = 0);
    std::optional<std::pair<OwnedMessage<T>, bool>> getLastUnreadAnswear();
}
```

```
template <typename T>
class Connection : public std::enable_shared_from_this<Connection<T>>
{
protected:
    const Owner owner_;
    std::thread threadRead_;
    std::thread threadWrite_;
    std::mutex mutexRead_;
    std::mutex mutexWrite_;
    std::condition_variable condVarRead_;
    std::condition_variable condVarWrite_;
    std::condition_variable& condVarUpdate_;
    boost::asio::io_context& context_;
    boost::asio::ip::tcp::socket socket_;
    common::ThreadSafePriorityQueue<OwnedMessage<T>>& incomingMessages_;
    common::ThreadSafePriorityQueue<Message<T>> outgoingMessages_;
    Message<T> incomingTemporaryMessage_;
    std::atomic_bool isReading_;
    std::atomic_bool isWriting_;
    std::atomic_bool shuttingDown_ = false;
    uint32_t id_;
    std::string ipAdress_;
    LOGGER("CONNECTION-UNDEFINED");

private:
    bool readData(std::vector<uint8_t>& vBuffer, size_t toRead);
    void readMessages();
    void writeMessages();
public:
    Owner getOwner() const;
    bool connectToServer(
        const boost::asio::ip::tcp::resolver::results_type& endpoints);
    bool connectToClient(uint32_t id);
    void disconnect();
    bool isConnected() const;
    void send(const Message<T>& msg);
}
```

```
template<typename T>
class Server
{
protected:
    common::ThreadSafePriorityQueue<OwnedMessage<T>> incomingMessagesQueue_;
    boost::asio::io_context context_;
    std::thread threadContext_;
    std::thread threadUpdate_;
    std::condition_variable condVarUpdate_;
    std::mutex mutexUpdate_;
    std::mutex mutexMessage_;
    boost::asio::ip::tcp::acceptor connectionAccepter_;
    common::ThreadSafeQueue<uint32_t> availableIds_;
    std::map<uint32_t, std::shared_ptr<Connection<T>>> connections_;
    std::atomic<bool> shuttingDown_ = false;
    LOGGER("SERVER");
private:
    void update();
public:
    bool start();
    void stop();
    void waitForClientConnection(); //ASYNC
    void messageClient(std::shared_ptr<Connection<T>> client,
        const Message<T>& msg);
protected:
    virtual bool onClientConnect(std::shared_ptr<Connection<T>> client);
    virtual void onClientDisconnect(std::shared_ptr<Connection<T>> client);
    virtual void onMessage(std::shared_ptr<Connection<T>> client,
        Message<T>& msg);
}
```

```cpp
template <typename T>
struct MessageHeader
{
    T type{};
    uint16_t id{};
    bool hasPriority = false;
    size_t size = 0;
};

template <typename T>
struct Message
{
    MessageHeader<T> header{};
    std::vector<uint8_t> body;

    size_t size() const;
    void clear();
    Message<T> clone();

    friend std::ostream& operator << (std::ostream& os,
        const Message<T>& msg);
    template<typename DataType>
    friend Message<T>& operator << (Message<T>& msg, const DataType& data)
    template<typename DataType>
    friend Message<T>& operator >> (Message<T>& msg, DataType& data)
}
```

4.3 CarDetector

To be able to connect to the road traffic cameras and count the number of incoming/outgoing vehicles wrappers over OpenCV library were made. The whole process works as follows: it starts detecting the moving objects inside a give frame, try to predict their next position and once they cross a imaginary lines inside the picture they are accounted if they were matched to be cars. To boost performance, the module runs on 4 threads: providing images from the camera, detecting moving objects, classifying objects as cars and displaying the bounding boxes. The last one can be disable, as it is just for illustrative purposes.

The car classification itself is not done by the given module due to current C++ Tensorflow - OpenCV outdated compatibility, but by the ObjectDetection-Server written in Python. The CarDetector module just acts like a client and sends the actual bytes of the cropped image of the detected moving objects to the server. It is important to note that if, in the near future, the compatibility issues will be fixed, it would server as a great performance upgrade to move this logic inside the module and remove any kind of process interactions.

4.4 ObjectDetectionServer

To be able to able to detect if cars were present we made use of Tenserflow Object Detection API 4.2. It is a powerful framework provided by Google's TensorFlow library that facilitates training, evaluating, and deploying object detection models. It offers a collection of pre-trained models and tools for building custom models to detect and localize objects in images and videos.

Figure 4.2: TensorFlow

Due to the fact that the model will be loaded by a microcontroller a SSD MobileNet model was trained by using a public dataset. SSD MobileNet is a combination of two popular deep learning architectures: Single Shot MultiBox Detector (SSD) and MobileNet.

SSD is an object detection algorithm that provides real-time object detection in images. It achieves this by predicting the bounding boxes and class labels of multiple objects in a single pass through a neural network. Unlike other object detection methods that require region proposal generation, SSD directly predicts object detections at different scales and aspect ratios.

MobileNet is a lightweight convolutional neural network architecture designed for efficient use on mobile and embedded devices. It utilizes depthwise separable convolutions, which separate the spatial filtering and channel-wise filtering operations, reducing the computational complexity and model size. MobileNet achieves a good balance between accuracy and efficiency, making it suitable for resource-constrained environments.

By combining SSD and MobileNet, the SSD MobileNet model achieves real-time object detection on mobile and embedded devices with limited computational resources. It provides a good trade-off between accuracy and efficiency, making it a popular choice for various applications such as autonomous vehicles, surveillance systems, and mobile applications requiring object detection capabilities.

The training was done with the use of tenserflow repository and it took about 15h on an RTX 3060 and the model is capable of detecting cars, street lights and other traffic related objects but it will be used just for detecting cars. The model speed is the most important thing here beeing able to evaluate 22 frames/s, as it's accuracy is not that great averaging 60%. The overcome the low accuracy any object that is more then 50% likely to be a car will be taken into account.

The model itself is loaded just once on a server, because it is a really time consuming operation, and used whenever a new valid message is recieved. Once the predictions are calculated, we remove the overlapping objects detected with the help of OpenCv NMSBoxes and return just the size of the remaining list of objects, not the bounding boxes themself, representing the amount of cars detected in the given image. To keep the consistency of the IPC implementation I imported the C++ message definitions and implemented the created a simple implementation of a server using socket package. To further improve the response time of the server, the request handling can be done in a asynchronous manner, having multiple threads working on different messages for the same client.

Related to the actual device that will run the server, it is better to just have them tied down to every proxy, due to the high resource need of loading the actual model that can not be achived on the microcontrollers.

4.5 Proxy

The proxy is a wrapper over the server implementation that assures the connection between the vechile trackers and the next junction main server they will encounter. Every single one is connected to their own database, that contains a list of junctions, proxys and the area covered by them.

They way it works is preaty straight forward, if a client querys the proxy for the next junction it searches the database. If it finds a suiting junction then it sends the connection data, otherwise it redirects the vehicle to the closeste proxy. This process repeats itself until it finally reaches one valid junction or the car is out of coverage area (for example if the car is somewhere on a ship in the middle of the sea there is no reason for us to connect it to a junction).

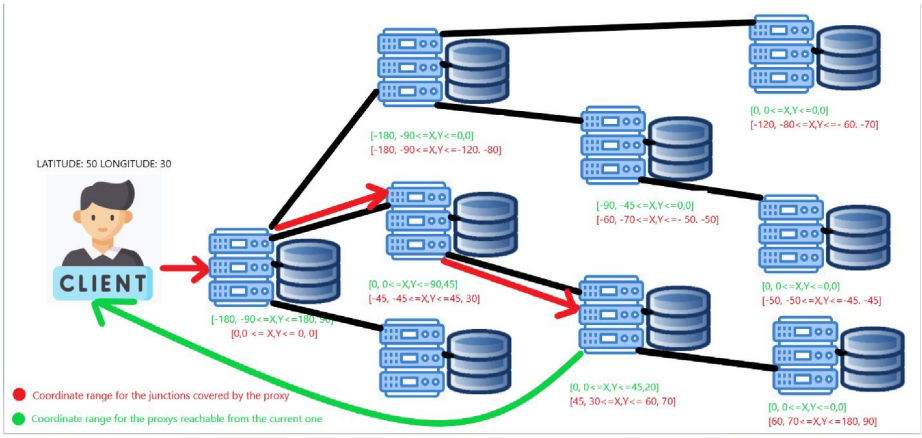


Figure 4.3: Proxy querys flow

The servers themself have a range of coordinate values for the covered junctions and a range of coordinate values for the known connected subproxys. The coordinate range of the know connected proxys lessens the more you go down the connection tree until an enpoint is reached, marked with range 0,0 <= >0,0. (Figure 4.3). If we still don't got an answear from the proxy containing a junction that means that we are out of reach and the client will just stop. Like this, the more servers we will have to spread our junction data, the faster the communication will be, as the client doesn't necessarily need to interogate the main server, it can interogate any of the servers.

4.6 JunctionMainServer

The executable is a wrapper over the server implementation that manages the traffic states. It keeps track of the incoming vehicles and times out each and every traffic state. The implementation itself is thought to be a state machine, with the states represented by the traffic states and the events being timer expire events. Each state has associated to it a timer that decreases normally or when an incoming vehicle is detected. Whenever we are in a given state, it's corresponding timer is frozen and afterwards reseted.

```
{
    "server": { "ip": "127.0.0.1", "port": 5000 },
    "maxWaitingTime": 2,
    "usingLeftLane": true,
    "lanes": {
        "top": "keyword1", "down": "keyword2",
        "left": "keyword3", "right": "keyword3"
    }
}
```

Both the timer for the green light and the ones attached to each lane dinamically update based on the traffic. To describe the way this is done we must first analise what increasing/decreasing the waiting time means for our system and also what could cause this scenario. What we would wish to achive is that every time a state ends X drivers have passed the junction. To do so, we control the amount of time a driver has to pass the junction. Also, the most important part is to minimize the amount of time those cars have to wait to pass the junction. Like so, we would want to increase/decrease the green light timer as well as the driver waiting timer attached to each lane. Figure 4.4 illustrates what negative effects playing arround with those timers would bring to our system.

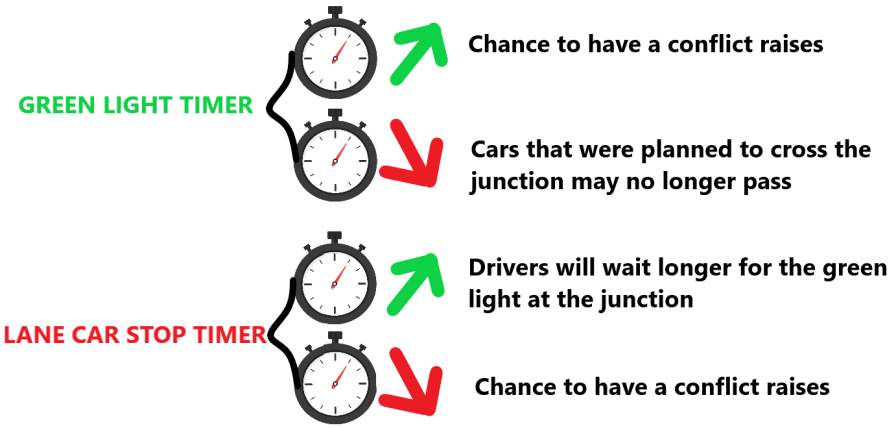


Figure 4.4: Timer Increase/Decrease Impact

To achive the desired traffic flow described above while minimizing the waiting time as much as we can we must be aware of 2 things: avoiding less cars passing the junction; avoiding conflicting states

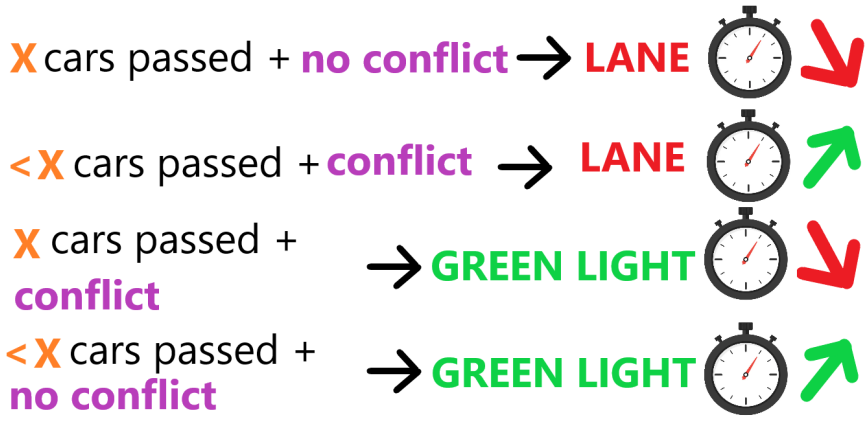


Figure 4.5: Faulty Scenarios Handling

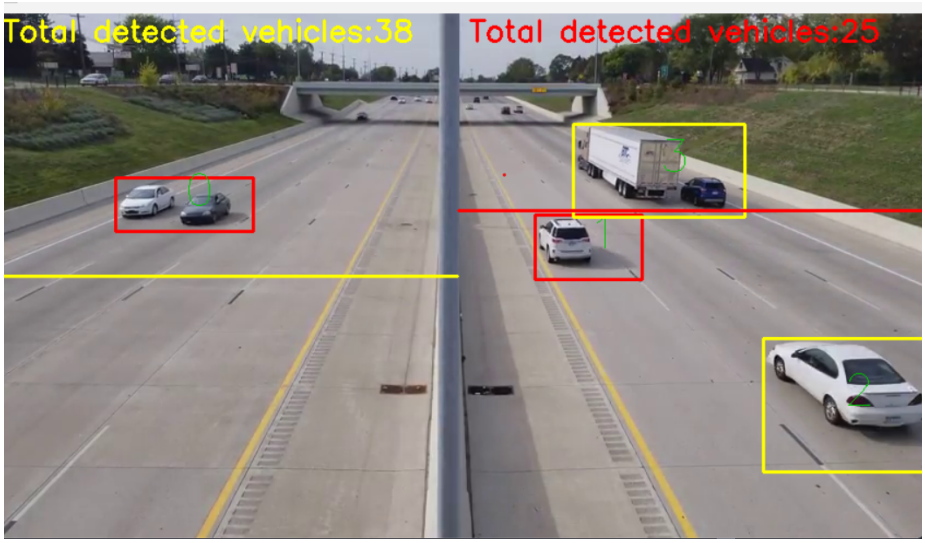
Figure 4.5 illustrates how we actually handle each combination of those scenarios. The way we determine if enough vehicles passed is by tracking the actual cars. Due to high complexity of image comparison, this number is based only on the VehicleTrackers input, so TrafficObserver client does not contribute to this process and significant decreases in waiting time can be seen only for the cars that are capable of using the VehicleTracker client.

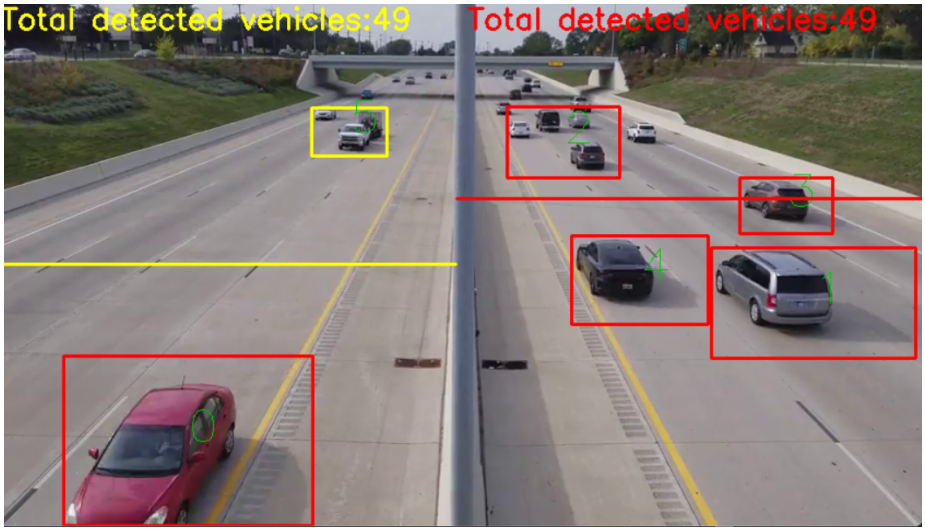
As for the way the lane timer decreases when a new vehicle is detected we will have a value that constanly increases the more cars are waiting. We also keep in mind an average of cars waiting to pass the junction and whenever the value is surpassed we start to drastically increase the amount of time decreased for the corresponding timer.

During the testing of the given module it was observed that traffic tends to keep it's normal flow, there is no jumping between traffic states unless a large inflow of cars is detected. Even if this happens, after the required jump transitions, traffic states will be normalized back to their initial flow definition.

4.7 TrafficObserver

The executable is a wrapper over the client implementation that combines cardetection functionalities. On startup it recieves a keyword for the junction to be able to determine what lane provided the incoming car message. To be able to run the executable itself you will need to provide a camera as well as a way to connect to the network, for example a zigbee. For testing porposes a video was provided instead of linking directly to a camera. 4.7





4.8 VehicleTracker

The executable is a wrapper over the client. On startup it tries loading a config file that contains an ordered list with the previous queried proxies. If not present, it will just interogate the main server. Once the application shuts down a config file will be saved with all the queried proxys, so it will almost always load a config file.

```
{
    "proxys": [
    { "ip_adress": "127.0.0.1","port": 6000},
    { "ip_adress": "127.0.0.5","port": 5000}]
}
```

The executable aims to be downloaded on each car system by the manufacturer and it requires a GPS module to be pressent as well as a wifi module. Messages start to be sent only when the vehicle itself is moving. To be able to determine this as well as it's heading GPS data is manually parsed, extracting latitude and longitude from NMEA GPGLL, GPGGA and GPRM inputs. For testing purposes data was already written inside a file and passed trough a pipe to the application. The data itself was taken from a public repository.

We first query the last visited proxy. If we are out of reach, we move up the proxy list until eventually we get valid junction connection data 4.6 or we get back to the root of the connection tree(the main server). If we have reached the root node, the process of searching for a junction without a config file restarts. Once we got the connection data we notify the junctions whenever we aproach/disconnect from it. When we have passed a junction the whole process restarts from the last queried proxy.

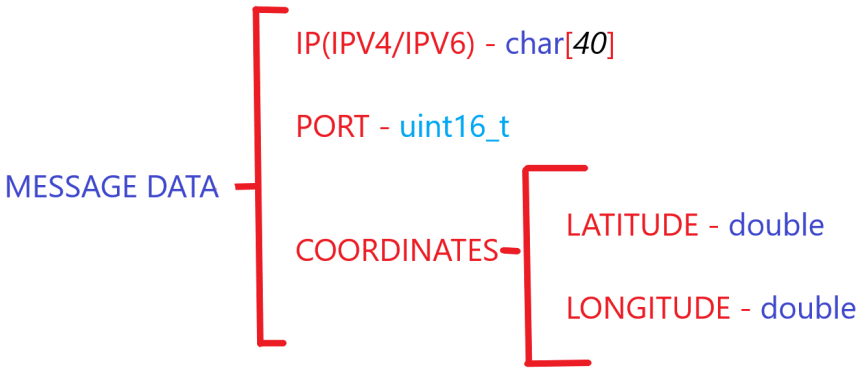


Figure 4.6: Proxy junction data reply

Chapter 5

Further Directions

Bibliography

[1] Dex R. Aleko and Soufiene Djahel. "An IoT Enabled Traffic Light Controllers Synchronization Method for Road Traffic Congestion Mitigation". In: *2019 IEEE International Smart Cities Conference (ISC2)*. IEEE, Oct. 2019. DOI: 10.1109/isc246665.2019.9071667.

[2] Akshay D. Deshmukh and Ulhas B. Shinde. "A low cost environment monitoring system using raspberry Pi and arduino with Zigbee". In: *2016 International Conference on Inventive Computation Technologies (ICICT)*. IEEE, Aug. 2016. DOI: 10.1109/inventive.2016.7830096.

[3] Junchen Jin, Xiaoliang Ma, and Iisakki Kosonen. "An intelligent control system for traffic lights with simulation-based evaluation". In: *Control Engineering Practice* 58 (Jan. 2017), pp. 24–33. DOI: 10.1016/j.conengprac.2016.09.009.

[4] Till Neudecker et al. "Feasibility of virtual traffic lights in non-line-of-sight environments". In: *VANET '12: Proceedings of the ninth ACM international workshop on Vehicular inter-networking, systems, and applications* (June 2012). DOI: 10.1145/2307888.2307907.

[5] Marcel Sheeny et al. "RADIATE: A Radar Dataset for Automotive Perception in Bad Weather". In: *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, May 2021. DOI: 10.1109/icra48506.2021.9562089.

[6] K.T.K. Teo, W.Y. Kow, and Y.K. Chin. "Optimization of Traffic Flow within an Urban Traffic Light Intersection with Genetic Algorithm". In: *2010 Second International Conference on Computational Intelligence, Modelling and Simulation*. IEEE, Sept. 2010. DOI: 10.1109/cimsim.2010.95.

[7] Ishu Tomar, Indu Sreedevi, and Neeta Pandey. "State-of-Art Review of Traffic Light Synchronization for Intelligent Vehicles: Current Status, Challenges, and Emerging Trends". In: *Electronics* 11.3 (Feb. 2022), p. 465. DOI: 10.3390/electronics11030465.

[8] R. Zhang et al. "Increasing Traffic Flows with DSRC Technology: Field Trials and Performance Evaluation". In: *IECON 2018 - 44th Annual Conference of the IEEE Industrial Electronics Society*. IEEE, Oct. 2018. DOI: 10.1109/iecon.2018.8591074.