Georg-August-Universität Göttingen
Faculty of Mathematics and Computer Science
Institute of Computer Science
Telematics Group
Sensorlab

A Car-to-X Communication Framework for autonomous cars with Wi-Fi Car Tracing and Traffic Light Control Optimizing

Winter 2016/2017

Chencheng Liang – University of Göttingen – 11603960
chencheng.liang@stud.uni-goettingen.de

Ishwarya Chandrasekaran – University of Hannover – 10006347
ishwarya.chandrasekaran@stud.uni-hannover.de

Abstract

This project demonstrates a Car-to-X communication framework for autonomous cars, in which car Wi-Fi localization, infrastructure control, and wireless communication between car and infrastructure is implemented. The Wi-Fi localization is a process that collects some useful information, such as IP address, Wi-Fi access point and calls Google API, which in turn processes this information and returns the longitude and latitude. The infrastructure control is considered to be an interface, which can receive the instruction coming from a server or any other source and execute that instruction in the corresponding physical infrastructure. The goal of this project is to build a communication framework that supports intelligent transportation, which helps to reduce the amount of fatal road accidents by providing traffic signal information to the car in advance and additionally control the infrastructure in case of an emergency situation. By informing the traffic status in advance we help car save fuel as well as manage the traffic condition dynamically depending on the density of cars at the traffic junction.

A simple Car-to-traffic light communication is shown in the video demo, in which car Wi-Fi localization remains the same as mentioned above and the infrastructure is considered as the traffic light. The communication between car and traffic light has two channels, one is server based communication and the other is a light weight messaging protocol called Message Queue Telemetry Transport (MQTT). In the MQTT communication, a message broker is implemented in the car itself rather than having a separate server.

1. Introduction

With the advancement in technology, it could be seen that everything around us is "smart" and connected with each other. The future belongs to autonomous cars and with this in mind, a communication framework which supports intelligent transportation is built. A framework for smart cars and smart infrastructure which could talk to each other to improve traffic efficiency and safety has been developed. It is a Car-to-X communication framework for car localization and infrastructure control. "X" could be any infrastructure for e.g. traffic light, garage or another vehicle, pedestrian etc. In this project "X" is considered to be traffic light.

The wireless communication framework has two implementations. First is using a UDP server, where the server is responsible for all tasks such as, receiving the location information from the car and generating instructions for traffic light based on this information and sending instructions to the traffic light. In the other implementation MQTT is used, which works based on publish-subscribe model using message broker. The MQTT message broker is implemented in the car itself; thereby the car can communicate with the traffic light directly and control the traffic light in case of emergency. Autonomous cars have gained a lot of attention in the recent years. In the future, it is believed that there will be more autonomous cars which are connected and could interact with each other where this framework could be used for Car-to-X communication

The objective of this project is to have the autonomous cars connected with the infrastructure in a smart way so that they could exchange information at ease. This not only helps the car to gain insight about the traffic situation and take decisions accordingly but also the traffic light can optimize the traffic flow dynamically depending on the number of cars at a traffic junction. Hence this data exchange between the car and traffic light has the following benefits (i) optimizes traffic flows (ii) increases road safety (iii) helps car to save fuel.

The rest of the report is organized as following: section 2 gives an overview of Car-to-X communication framework. Section 3 describes the methodology and design of the framework. Section 4 describes implementation with the explanation of the demo and results. Section 5 provides evaluation of the framework and section 6 describes conclusion and future works.

2. Overview of our Framework

This framework is built considering three use cases. First scenario is communication between normal car and traffic light. Second is communication between special car (without siren) and traffic light. The third scenario is communication between special car (with siren) and traffic light. The

difference between the second use case and the third use is that, different wireless communication mechanisms are used. For the use case, special car (without siren), UDP server communication mechanism is implemented. For the other use case, special car (with siren), MQTT messaging protocol is implemented for data exchange between the car and traffic light. MQTT uses a message broker for data exchange. In this project, the message broker is implemented in the special car rather than installing in a separate node. The reason being special car (with siren) is considered to be the highest priority case and it requires instant transmission of data with high reliability over the wireless networks. So, when the broker is implemented in the special car itself, it facilitates real-time communication in an efficient manner.

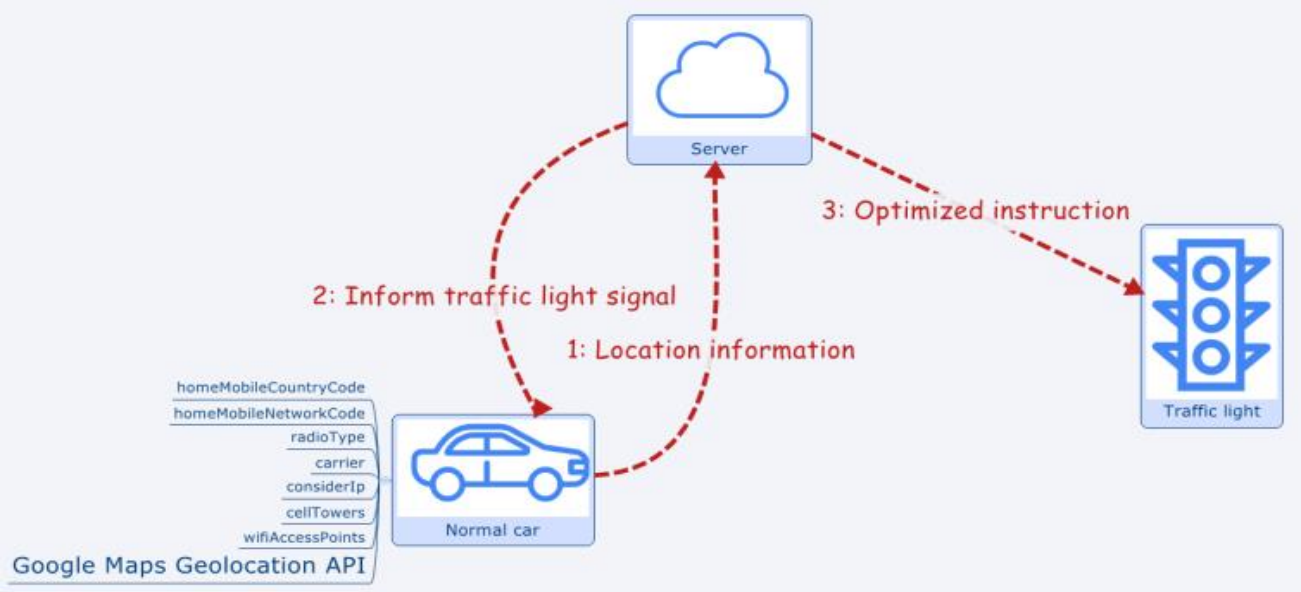

Figure 1. Normal car communication diagram

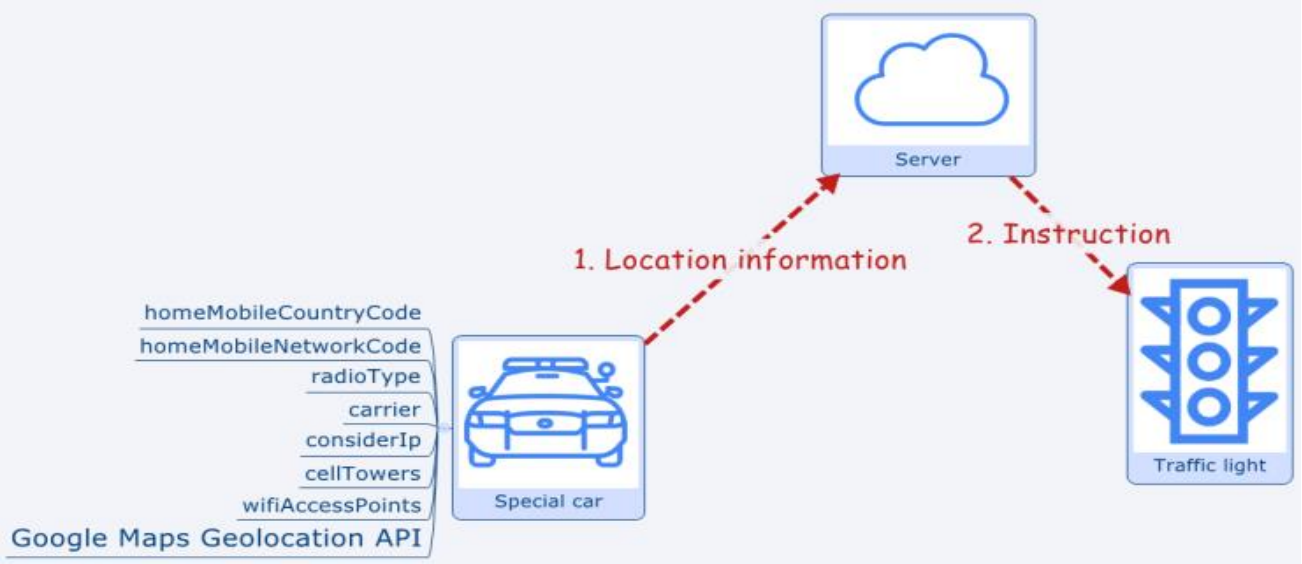

Figure 2. Special car (without siren) communication diagram

UDP Server based communication for normal car is shown in Figure 1. The communication between the normal car and traffic light is described as follows: Normal car runs the Wi-Fi localization process which collects the following information from the car:

I. homeMobileCountryCode (MCC)
II. homeMobileNetworkCode (MNC)
III. radioType
IV. carrier
V. IPAddress
VI. cellTower
VII. Wi-FiAccessPoints

A request is sent from the car to Google Maps Geolocation API with this information, which in turn replies with the latitude and longitude information of the car along with the accuracy of this information. A detailed explanation of this process is provided in the implementation section. After receiving its location information, the normal car sends it to the server. The server receives this information and generates a new traffic instruction according to the recent location information received and sends this new instruction (e.g turn green signal ON in the direction of the car) to the nearest traffic light. This traffic light signal is also sent to the normal car from the server. Two directions are considered to decide which signal (red or green) to turn on. Let's say the direction in which the car is travelling is x-axis and the direction perpendicular to the axis of the travelling car is y-axis. So, depending upon the location and direction information from the car, the server knows in which direction the car is coming and sets the traffic instruction to be green in that direction (axis). The optimized traffic instruction is calculated based on the number of cars in the x and y axis.

UDP Server based communication for special car (without siren) is shown in Figure 2. Special cars such as police car or ambulance obtains its location from the Google Maps Geolocation API as described in the above paragraph and send its location information to server. Since this is a special car, the server generates a green signal instruction and sends this instruction to the nearest traffic light. So, every traffic light nearby the special car will turn green light for it.

The third use case considered is special car (with siren). This is shown in Figure 3. For this use case, MQTT messaging protocol is used for exchanging messages between special car and traffic light. MQTT is a lightweight publish-subscribe protocol used on top of TCP/IP. Using MQTT nodes can subscribe or publish on a message topic. The communication between the publisher and the subscriber nodes happen via MQTT message broker. For example, nodes can subscribe on a message topic to the broker. Whenever another node publishes a message on the same topic, this message is delivered to

subscribers by the broker [1]. The details of how the subscribe-publish mechanism works is described in the implementation section.

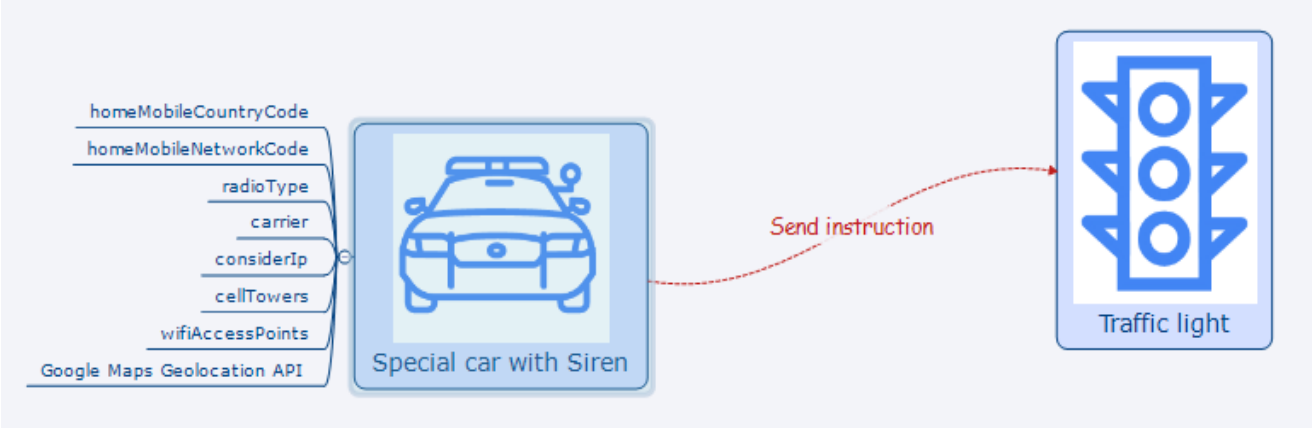

Figure 3. Special car (with siren) communication diagram

3. Methodology and Design

Sequential diagrams have been used to explain the sequence of steps executed in this project.

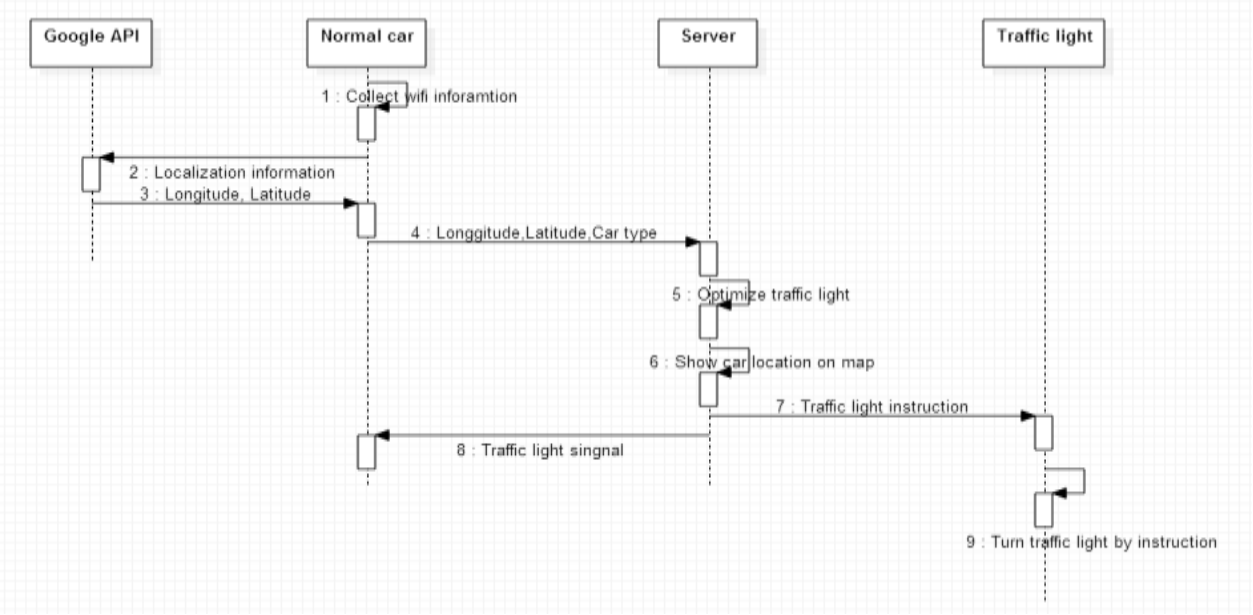

Figure 4. Normal cars sequential diagram

Figure 4 shows how the interaction between normal car and traffic light is achieved. The sequence of the interaction is described as follows:

1: The normal car carries a Raspberry Pi (RPi) which collects as much information as possible which is required for localization as mentioned in section 2.

2: Google Geolocation API is called from the normal car to process this information.

3: The Google Geolocation API returns a location which consists of longitude and latitude.

4: The RPi on the car sends this longitude and latitude along with its car type (normal) to the server which has a fixed IP address.

5: After receiving the longitude and latitude of the car, the server runs a simple algorithm to calculate optimized traffic light instruction and sends it to the nearest traffic light. The server runs an algorithm to decide the nearest traffic light for the car. The new traffic light information is also sent to the car. These two algorithms are explained in implementation section 4.3.1. Server based communication demo.

6: This step is optional. When location information is received from the car, it can be displayed in Google Maps along with the location of Traffic light. This needs a chrome driver and chrome browser. A simple result obtained from the demo is shown as a Google Map in Figure 5. The symbol T (green point shown in Figure 5) denotes traffic light, which is a RPi placed in the sensor lab in Göttingen and the symbol C (red point shown in Figure 5) denotes car, which is another RPi placed in a different location near sensor lab.

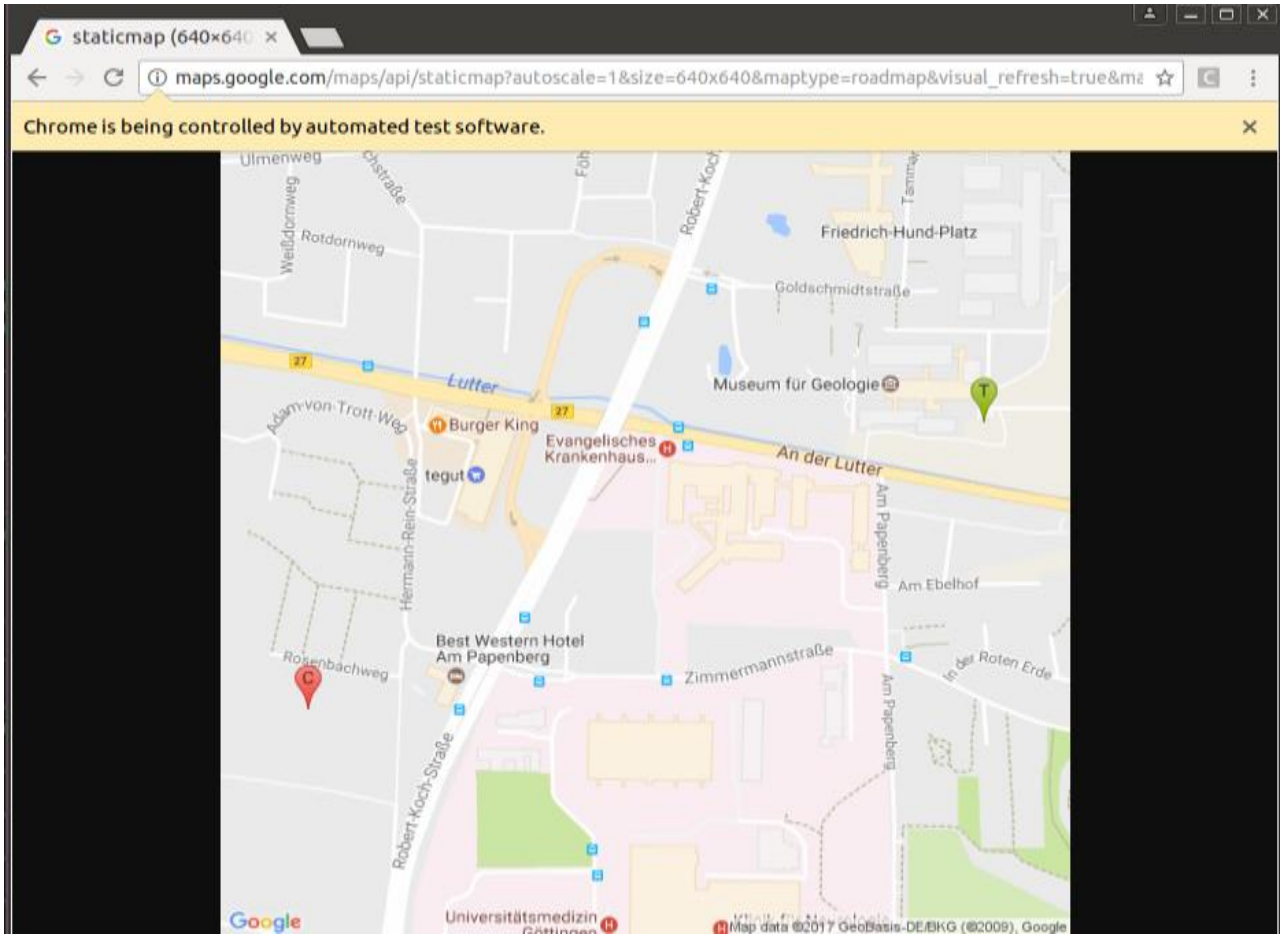

Figure 5. Position of the car and the traffic light in demo

7: After the traffic instruction is calculated from step 5, it is sent to the traffic light by the server, which

will be the new signal.

8: The server sends the new traffic light status to the car. Depending on this traffic status, the car can adjust its speed when encountering a red traffic signal or green traffic signal. This improves traffic flow and reduces emissions.

9: The traffic light receives the calculated instruction from server. If the traffic instruction is same as the current signal state then the traffic light will not change its state. If the received traffic instruction is different from current signal state, the traffic light will change its state according to the instruction.

Figure 6 shows the interaction between special car (without siren) and traffic light. The sequence of the interaction is described as follows:

1: The special cars such as police car or ambulance carry RPi. First the RPi collects the information for localization. This is same as the first step shown in Figure 4.

2, 3 and 4: These steps are same as steps 2, 3, 4 shown in Figure 4.

5: It is same as step 6 shown in Figure 4.

6: If the server receives the information from special car i.e if carType = special, then the server generates an instruction to turn the green light ON in the direction of the special car.

7: The server sends the calculated traffic instruction to the traffic light.

8: After receiving the instruction, if the traffic light is already in green it will remain in the same state if not it will change the signal state to green for the special car.

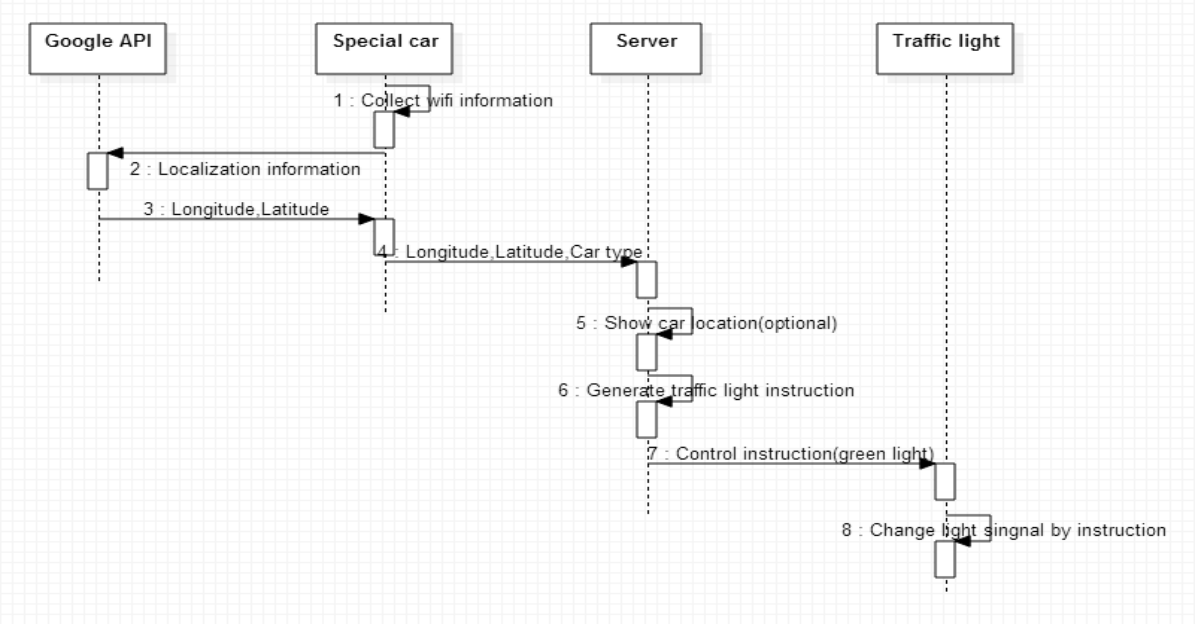

Figure 6. Special car (without siren) sequential diagram

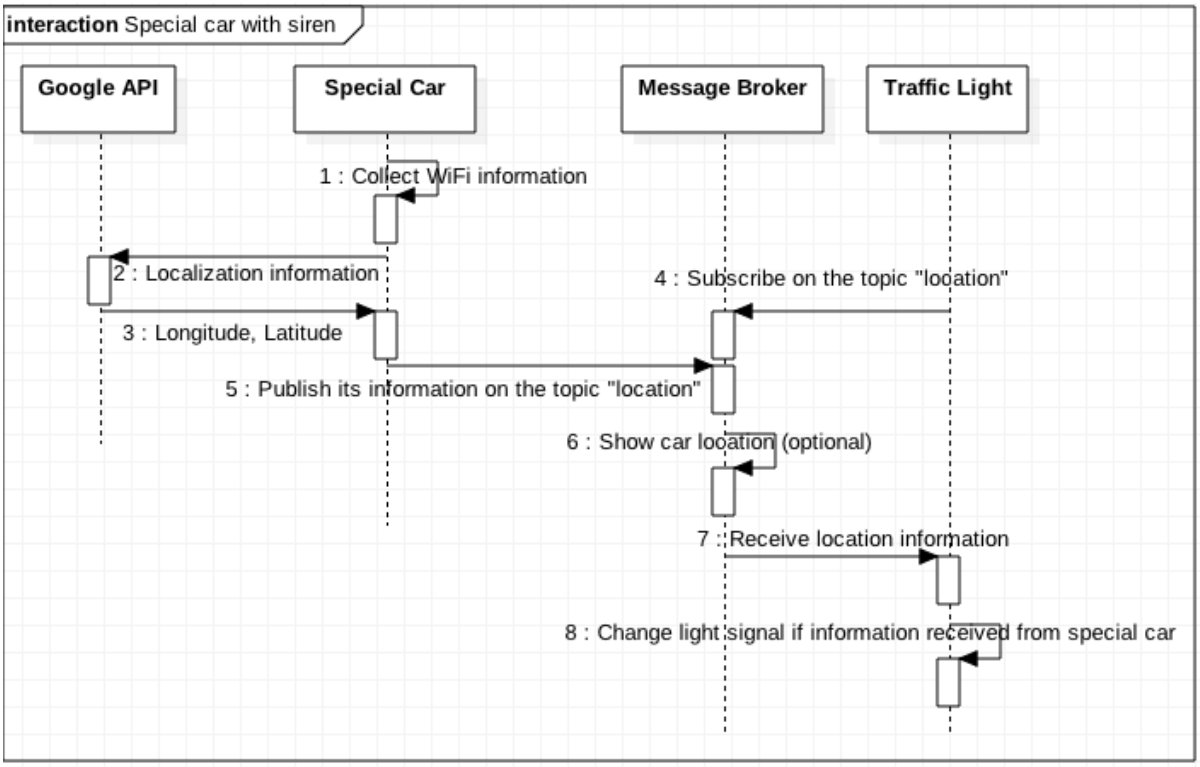

Figure 7. Special car (with siren) sequential diagram

Figure 7 shows the interaction between special car (with siren) and traffic light. The sequence of the interaction is described as follows:

1: The special cars such as police car ambulance with siren carry RPi. Instead of using a separate server for processing the location information, message broker is used as a central point of communication. First the RPi collects information for localization. This step is same as the first step shown in Figure 4.

2, 3: These steps are same as steps 2, 3 shown in Figure 4.

4: The traffic light which likes to receive location information from the special car subscribes on a message topic say “location” to the broker.

5: When the special car receives its location from Google Maps API, it publishes its location information on the topic “location” to the message broker.

6: It is same as step 6 shown in Figure 4.

7: Since the traffic light has subscribed on the message topic “location”, the message broker delivers the location information from special car to traffic light.

8: The traffic light receives this information from message broker. If the traffic light identifies that this information is from a special car (with siren), it changes its signal state to green and be in this state for few seconds. If the signal is already in green state, it maintains in that state for few more seconds.

4. Implementation

In this section, the design implementation and the demo of the project along with the results is discussed. This project is designed for multiple cars and multiple traffic lights, but for the demo purpose a normal car, a special car and a server is used to test the application. By running this demo in field, it is shown that this project is working in real scenario rather than only on local virtual machines and node simulators.

4.1 Hardware preparation

According to the design, two Raspberry Pi basic sets and a server is required for building the application. A Raspberry Pi set includes one RPi 3, a SD card, a HDMI cable, a power supply, a USB dongle. One RPi 3 is deployed in a car and another is used as an emulator for traffic light, which can be deployed at a traffic junction on the road. The SD card is the storage used for RPi 3. It contains operating system and can store files that are generated or download. The power for RPi 3 can be fed from power bank or from 230 Volt mains. The USB dongle is used for connecting RPi 3 to a Wi-Fi network. A monitor can be connected to RPi through HDMI cable, to view the RPi GUI for programming and testing the demo. In the real scenario, monitor and HDMI cable are not necessary. A virtual machine or real machine can be used as a server with Ubuntu operating system running on it. For emulating the traffic light signal, LEDs connected to bread board are used to demonstrate the traffic light signal state. Resistors are used to connect LED to ground, to provide constant supply of voltage to the LED.

4.2 Software preparation and deployment

Raspbian operating system (OS) for RPi is downloaded on the SD card and installed [2]. A Linux server virtual machine (VM) with Ubuntu OS running is built. The server IP should be fixed. Since there is no fixed IP available, the server IP is provided as input during the code execution in the demo.

Prerequisites for executing the framework:

Python is used for the framework development. Other installation required to run the project are as follows:

I. pip
II. selenium
III. uuid

IV. paho MQTT

V. mosquito message broker

Installation steps:

I. To install pip in Raspbian and linux server, issue the command “sudo apt-get install python3-pip”.

II. To install selenium and uuid issue commands “sudo pip install selenium” and “sudo pip install uuid”.

III. For installing MQTT protocol on the traffic light RPi and special car RPi, issue the command “pip install paho-mqtt”. This command installs paho client for MQTT which can be used in python environment.

IV. The “mosquito” message broker is installed in the special car RPi with the command “sudo apt-get install mosquito”.

The full source code can be found in the GitHub repository link [3]. All installation commands are specified in “install.sh” in the GitHub repository. By executing this bash script all the prerequisites necessary to execute the application will be installed. Other thing to be noted is, to get access of Google Geolocation API, it is required to apply for a key (key can be acquired from the link [4]). For showing the location of car and traffic light in Google Maps, chrome driver is used. Chrome driver can be downloaded from the link [5]. As this file is not large, it is included it in the code file. A Google browser is also required to run the map.

For the server based communication, the files “normal_car.py”, “special_car.py” is loaded in one RPi and “TrafficLight.py” is loaded in another RPi. The “server.py” should be loaded on Linux server VM.

For MQTT based communication, the files “special_car_siren.py”, and “mqttPublishMultiple.py” are loaded in the special car RPi. The files, “mqttSubscribeSimple.py” and “TrafficLight_mqtt.py” are loaded in the Traffic light RPi.

4.3 Demo explanation and results

The results of UDP server based communication have been shown in the first demo video. Section 4.3.1 explains this demo. As the results were already shown in the demo video, screen shots are not provided in this section.

4.3.1 Server based communication demo

The normal car scenario execution:

1. TrafficLight.py code is executed on a Raspberry Pi. In TrafficLight.py a socket is used to receive a UDP packet at 8080 port. The packet contains the instruction to control the traffic light.

2. server.py code is executed on Linux system. In server, a socket is used to receive UDP packet from port 2333. Since the car type is normal in this scenario, when the server code is executed, first it runs an algorithm to decide the nearest traffic light for the car. The algorithm is rather simple; it uses the received longitude and latitude of the car and subtracts it with longitude and latitude of all traffic lights in the surrounding area. Then the latitude and longitude value of each traffic light is added. Traffic light with the smallest absolute value is considered the nearest traffic light to the normal car. For example, if there are two traffic lights with their location information (latitude, longitude), say traffic light 1 (55, 56) and traffic light 2 (31, 45) and the received car location is (50, 50), then the result of subtraction for traffic light 1 and 2 are (5, 6) and (-19, -5) respectively. The absolute value for the sum of latitude and longitude for traffic lights 1 and 2 are calculated as 11 and 24 respectively. So, the nearest traffic light for the car in this case would be traffic light 1.

Next the server runs an algorithm to generate an optimized traffic light signal. In this algorithm, the number of cars on the direction of the car (ie. x axis) and in the perpendicular direction of the car (y axis) is calculated. Depending on the number of cars in both directions, a new traffic instruction is generated by the server. For example, if the x axis has more cars than y axis, then the instruction to turn the green light “ON” on the x-axis is set and vice versa.

After the new traffic instruction is generated, server sends a UDP packet containing traffic light instruction to the nearest traffic light using the socket with port 8080. The server also sends a UDP packet containing the current traffic light status to normal car using socket with port 9090.

3. normal_car.py code is executed on another Raspberry pi. In normal_car.py, a socket is used to send the collected location information as a UDP packet (longitude, latitude, car type, and car IP address) to server with port 2333. The normal car receives UDP packet from server with port 9090 with the current traffic light signal.

The normal_car.py file which contains the code for Wi-Fi localization relies on Google Maps Geolocation API. Initially the location information is collected and put into a JSON format as shown below:

```
{
 "homeMobileCountryCode": 310,
 "homeMobileNetworkCode": 410,
 "radioType": "gsm",
 "carrier": "Vodafone",
 "considerIp": "true",
 "cellTowers": [
     "cellId": 42,
    "locationAreaCode": 415,
    "mobileCountryCode": 310,
    "mobileNetworkCode": 410,
    "age": 0,
    "signalStrength": -60,
    "timingAdvance": 15
 ],
 "wifiAccessPoints": [
  "macAddress": "00:25:9c:cf:1c:ac",
 "signalStrength": -43,
 "age": 0,
 "channel": 11,
 "signalToNoiseRatio": 0
 ]
}
```


This information is sent to the url, https://www.googleapis.com/geolocation/v1/geolocate?key= API_KEY using "urllib2.Request" method. The "urllib2.Request" returns a URL as a response. "urllib2.urlopen" method is used to open the returned URL. The value returned from urllib2.urlopen is in a JSON format:


```
{
 "location": {
   "lat": 51.0,
   "lng": -0.1
 },
 "accuracy": 1200.4
}
```


In the above description, "lat: 51.0" denotes latitude, "lng: -0.1" denotes longitude and "accuracy": 1200.4 denotes the location radius, which means the real location will be in a circle with the central coordinates (51,-0.1) and radius 1200.4 meters. More detail on how to use Google Maps Geolocation API for localization can be found in the link [6].

The special car scenario execution:

1. TrafficLight.py code is executed on a Raspberry Pi, this is the same as normal car scenario.

2. Run server.py code is executed on Linux server. After server receives the packet from the special car (UDP packet, port 2333), it decides the nearest traffic light for the car which is same as described in

normal car scenario. Then the server checks the car type, since this is a special car, instead of executing the algorithm to get optimized traffic light signal instruction, the server directly sends a green light signal to the nearest traffic light (UDP packet, port 8080). If the car type is special then it is not required for the server to send the traffic light signal status to the special car, because the algorithm is designed in a way that if the car type is special then all traffic lights nearby the special car will turn green on the same direction as that of special car.

3. special_car.py code is executed on another Raspberry Pi. This is almost the same as the normal car scenario but the only difference is that car type parameter is 'special'

4.3.2 MQTT based communication demo

The steps for executing the scenario special car with siren based on the MQTT communication is as follows:

1. First trafficLight_mqtt.py code is executed on one RPi. Traffic Light is the subscriber which subscribes on a topic "/location" to the broker. "paho.mqtt.subscribe" library should be imported in this code, if the Traffic light wishes to subscribe for location information. Subscribe command is as follows: *subscribe.simple(topic_name,broker_IP,message_count).* Now the Traffic Light keeps waiting until it receives a message on the topic "/location". In the demo 2 shown in video, when the traffic light keeps waiting to retrieve the data, the signal state is shown as RED.

2. LED is connected to RPi GPIO. In the demo, it is explained that when the Traffic Light subscribes for the location data, the signal state changes to green.

3. Next ""special_car_siren.py" code is executed on another RPi. This acts as the publisher which publishes its location data retrieved from Google Geolocation API to the message broker with the message topic "/location". "paho.mqtt.publish" library should be imported in this code, if the special car wants to publish its location information to the broker. Publish command is as follows: *publish.multiple(messages_to_be_published,broker_IP).*

4. Once the location data is published from the special car, the Traffic Light signal state changes to GREEN (this could be seen in the demo 2 shown in the video) and the location information is received by the Traffic Light. After few seconds, the traffic light changes to RED state, assuming that the special car has crossed the traffic junction and the program stops execution.

4.4 The problems encountered in practice

One of the problems encountered during the demo is to get accurate location information. The

latitude and the longitude information were obtained with an accuracy of around 1200 meters radius. In the future work this information can be made more accurate.

Another problem faced during the development of the framework is the network and subnet issue. Since Göttingen network is used for demonstration, there is no fixed IP address available for the server and hardware devices used in the project always needs to be connected in the same subnet. For convenience in the demo, server IP address is obtained as an input parameter. For example, the server code is executed with the command "python special.py 10.10.13.157". But if a fixed IP address is available for the server, it is not necessary to input the IP address for execution.

The three devices used in the demo are connected in one subnet, which is provided by campus, called GeoMobile/GuestOnCampus. Since the devices are connected under one subnet, they can communicate with each other.

5. Evaluation

One of the strength of this project is that indoor localization is achieved, which overcomes the drawback of GPS localization mechanism which requires an open space and sometimes requires good weather for calculating coordinates. Another important thing to be noted is that, in this project not just a Wi-Fi localization and traffic light control application is developed but a localization-based node control framework is built, which exposes programming interfaces on the server and the traffic light. These interfaces could be controlled to handle any type of application. For example, the traffic light could be replaced with another infrastructure and different set of instructions can be executed to satisfy the requirements. A possible application that could be developed with this framework is as follows: RPi can be installed in the car garage of a person's house, so whenever the person comes with his car to the garage, the server recognizes it and sends instruction to the RPi in the garage which in turn opens the garage door. Since the code is written in Python, any other hardware apart from RPi which supports Python can run this project. Considering the MQTT based communication is lightweight and is well suited for devices with limited power or in the areas where the network connectivity is not reliable.

One design drawback for the server based communication framework is that Raspberry Pi is installed in the car to get access to Google API, to retrieve the longitude and latitude which then sends this information to the server. It would be efficient if these processing can be offloaded to the server, so the tracking device on the car or the infrastructure could be more energy-efficient.

6. Conclusion and future works

A Car-to-X communication framework is built and implemented with a Car-to-Traffic light application. Apart from installing the RPi on car and traffic light, as per our framework, the RPi could be installed on any mobile object for smart communication. In the future, this project could be extended to include Car-to-Pedestrian and later on to Car-to-Car communication as well. Currently the client-server communication is built with UDP but this has a lot of network problems caused by NAT, firewall, etc. So in the future, it is planned to build a Browser-Server (B/S) system, so that the car only needs to post/fetch its information to/from the web server. With this B/S structure, the tracking device in the car could be any hardware which provides tracking function. This device is only required to put its location information in the web server. Any infrastructure which needs to receive this information needs to have access to the web server, so that it can obtain the information. Also, currently there is no security mechanism implemented in this framework. In the future, security mechanism could include user authorization and Transport Layer Security (TLS) and Secure Sockets Layer (SSL) implementation.

7. References

1. https://en.wikipedia.org/wiki/MQTT

2. https://www.raspberrypi.org/downloads/raspbian/

3. https://github.com/lcckkkhaha/A-Car-to-X-Communication-Framework-for-autonomous-cars-with-Wi-Fi-Car-Tracing-and-Traffic-Light-Cont

4. https://developers.google.com/maps/documentation/geolocation/intro

5. https://sites.google.com/a/chromium.org/chromedriver/

6. https://developers.google.com/maps/documentation/geolocation/intro#cell_tower_object