Universidade de Aveiro

Redes e Sistemas Autónomos

Project Report

João Amaral (98373), João Viegas (98372)

Departamento de Eletrónica, Telecomunicações e Informática

June 16, 2023

Contents

List of Figures

Chapter 1

Abstract

The advent of Intelligent Transportation Systems (ITS) presents a unique opportunity to re-imagine infrastructure for greater safety and energy efficiency. Our project, the Smart Adaptive Lighting System, exploits this potential by leveraging Vehicle-to-Infrastructure (V2I) communication technology. We have devised a system that uses a limited number of Road-Side Units (RSUs), integrated into selected smart lamp posts, to process Cooperative Awareness Messages (CAMs) from vehicles' On-Board Units (OBUs), that dynamically adjusts road lighting based on vehicle presence, speed, and proximity. Communication is facilitated via the ETSI C-ITS protocol using Vanetza extended with MQTT and JSON capabilities, a network stack tailored for ITS, with the vehicles communicating their location and speed through Cooperative Awareness Messages (CAMs). Some lamp posts, processing the CAMs through integrated RSUs, calculate an optimal light intensity for each approaching vehicle and relay this information to neighboring posts via a novel communication medium, the Light Support Messages (LSM). Our system thereby delivers a seamless and adaptive lighting experience for drivers, contributing to enhanced road safety, while also significantly reducing energy consumption and infrastructure costs.

Chapter 2

Introduction

Traditional road lighting systems often function at a constant high intensity throughout the night, irrespective of traffic volume or vehicle presence. This approach not only leads to significant energy waste, but also contributes to the increasing problem of light pollution in our urban areas. It's clear that this area of urban infrastructure is ripe for an innovative, intelligent solution.

Our response to this challenge is the development of the Smart Adaptive Lighting System. Our system employs Vehicle-to-Infrastructure (V2I) communication technology to dynamically adjust road lighting in response to real-time traffic conditions. By integrating Road-Side Units (RSUs) into selected smart lamp posts and using On-Board Units (OBUs) in vehicles, we're able to process and respond to live traffic data, enabling the adaptive lighting system.

A mixture of RSU and non-RSU equipped smart lamp posts is utilized in our system. The RSUs process vehicle data communicated via Cooperative Awareness Messages (CAMs), calculate optimal light intensity, and relay this information to all nearby posts using our novel Light Support Messages (LSM). The streamlined design reduces infrastructure costs without compromising the functionality or effectiveness of the system.

V2I is facilitated via the ETSI C-ITS protocol using Vanetza, a network stack designed specifically for ITS. By harnessing these advanced technologies and protocols, we've designed a system that provides a safer and more efficient driving experience, all while contributing to energy conservation and reducing light pollution. As such, the Smart Adaptive Lighting System embodies the promise of smart city initiatives and the transformative power of ITS.

Chapter 3

Objectives of the work

The primary objective of our project is to design and implement a smart adaptive lighting system that adjusts road lighting based on real-time traffic conditions. This includes developing a method to gather vehicle data, process it, and translate it into optimal light intensities.

An integral part of our system is the use of V2I communication technology. Our goal is to effectively leverage this technology, utilizing the ETSI C-ITS protocol through the NAP-Vanetza network stack, for seamless transmission and processing of information.

In the interest of cost-efficiency, another objective is to create a system that works effectively with a mix of RSU-equipped and internet-only lamp posts (these "internet-only" lamp posts only need a small computational unit to process data from an active connection to the RSUs network). This involves designing a communication model where LSMs can be relayed effectively to both types of lamp posts, with internet-only posts able to select the highest suggested light intensity.

By using smart adaptive lighting, we aim to significantly reduce energy consumption associated with road lighting. Additionally, by preventing unnecessary high-intensity lighting, we hope to contribute to the reduction of light pollution.

Chapter 4

Architecture

4.1 Overview

The architecture of our Smart Adaptive Lighting System plays a crucial role in enabling its innovative functions and optimizing its performance.

Based on the NAP-Vanetza docker image, it consists of three main components: On-Board Units (OBUs) in vehicles, Road-Side Units (RSUs) integrated into selected smart lamp posts, and additional Smart Lamp Posts. Each component plays a unique role in the system, contributing to a comprehensive, intelligent solution for dynamic road lighting. In the following section, we delve into the specifics of this architecture, exploring the role and function of each component, their interactions, and the overarching communication protocols employed.

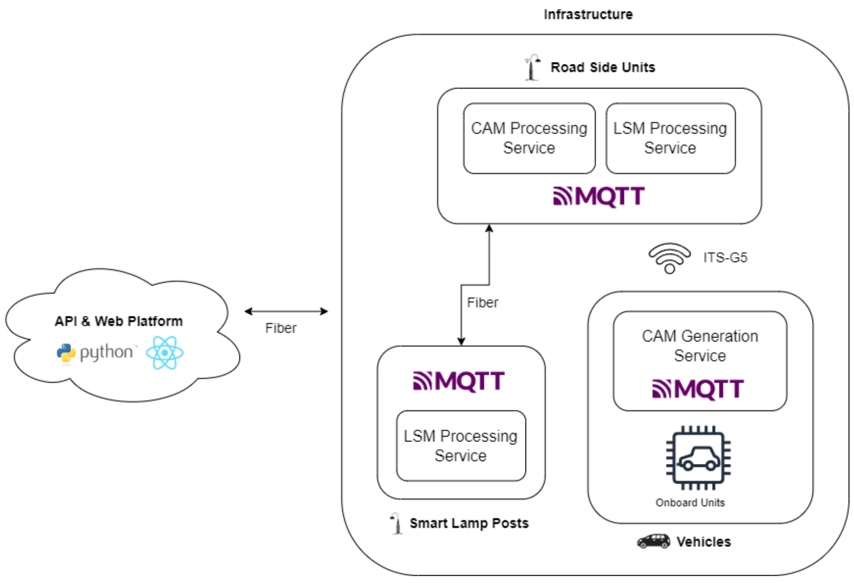


Figure 4.1: System architecture

4.2 Component Description

We now turn our focus to the distinct elements that constitute the architecture of the Smart Adaptive Lighting System. Each component, whether it's an On-Board Unit, a Road-Side Unit, or a Smart Lamp Post, plays a vital role in enabling the system's dynamic lighting control.

The OBUs are responsible for transmitting CAM messages that contain important and relevant information like their current position as latitude and longitude coordinates and their speed. On the other hand, the RSUs are able to receive the CAM messages transmitted by the OBUs, process them and calculate the correct light intensity values for themselves as well as for other posts nearby them. Then those values are sent via LSM messages to all the corresponding posts. The Smart Lamp Posts are devices that only need to receive these LSM messages, process them and adapt their light intensity accordingly.

Lastly, we developed an API to capture all the messages transmitted, process them and return the information in specific endpoint to serve a front-end application that displays the full environment in detail and real-time.

4.3 Communication Interactions

The strength of our system lies in the successful interaction between its components, all working harmoniously to deliver optimal lighting conditions. The flow of information from OBUs to RSUs, and further to the lamp posts, orchestrates the intelligent response of our system.

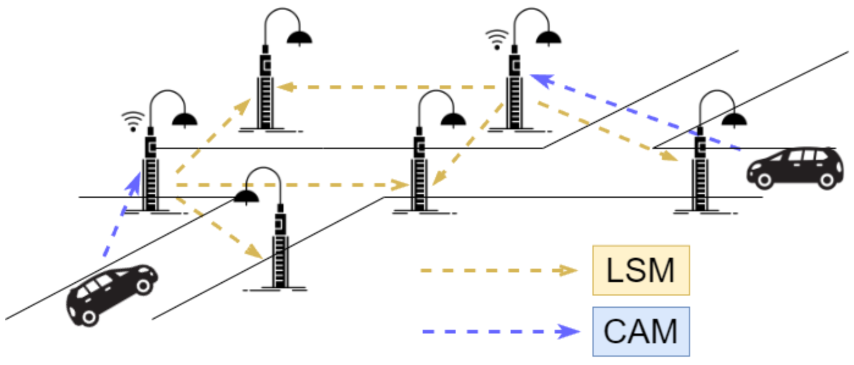


Figure 4.2: Communication schematics

The OBUs transmit CAM messages using their integrated MQTT broker, at a pre-defined frequency of 10Hz, publishing the corresponding JSON to the topic *vanetza/in/cam*. On the other hand, the RSUs are able to receive the CAM messages transmitted by the OBUs by subscribing to the topic *vanetza/out/cam*, also on their internal broker.

However, the LSM messages are published and subscribed in an external MQTT broker, accessible by the RSUs and Smart Lamp Posts network. For this particular project, we used the our computer running a Mosquitto MQTT service deployed in the Docker's VLAN that the Vanetza containers use.

Chapter 5

Implementation

5.1 OBUs Simulation

In this section, we delve into the implementation details of our project, with a specific focus on the creation of the OBU simulation using Python. The simulation forms the cornerstone of our system, effectively mimicking real-world driving conditions to test and demonstrate the functionality of the Smart Adaptive Lighting System.

Algorithm: Given two points, a specific speed and a transmission rate of 10Hz, the following function returns a list of coordinates between those points.
Input: StartPoint (lat1, long1), EndPoint(lat2,long2), Speed
Output: List of coordinates of that path

```
def get_coordinates(lat1, lon1, lat2, lon2, speed):
    distance_meters = geodesic(start, end).kilometers * 1000
    speedms = speed * 0.277778
    seconds = distance_meters / speedms
    N = 10
    #step size in meters
    step = distance_meters / (seconds * N)
    bearing = calculate_initial_compass_bearing((lat1, lon1), (
                                            lat2, lon2))
    coordinates = []
    for i in range(samples+1) do
        step_distance = step * i
        step_point = geodesic(meters=step_distance).destination
                                            (point=start, bearing=
                                            bearing)
        coordinates.append((round(step_point.latitude, 6),
                                            round(step_point.
                                            longitude, 6)))
    return coordinates
```

We then calculated the corresponding lists of coordinates for the streets we wanted our vehicles to travel through and joined them, to create the desired paths. For example:

```
A25_1 = [(40.636800, -8.667729),(40.639178, -8.664639),100] #
                                A25 first part at 100km/h
A25_2 = [(40.639178, -8.664639),(40.642825, -8.659768),110] #
                                A25 second part at 110km/h
A25_3 = [(40.642825, -8.659768),(40.646065, -8.654425),120] #
                                A25 third part at 120km/h

#get the coordinates of those three segments
A25_1_street = coordinates_to_dict(get_coordinates(A25_1[0][0],
                                 A25_1[0][1], A25_1[1][0],
                                A25_1[1][1], A25_1[2]))
A25_2_street = coordinates_to_dict(get_coordinates(A25_2[0][0],
                                 A25_2[0][1], A25_2[1][0],
                                A25_2[1][1], A25_2[2]))
A25_3_street = coordinates_to_dict(get_coordinates(A25_3[0][0],
                                 A25_3[0][1], A25_3[1][0],
                                A25_3[1][1], A25_3[2]))

#create the path itself and the speed of each segment
path = [A25_1_street, A25_2_street, A25_3_street]
speed = [100, 110, 120]

#simulate the OBU travelling through those coordinates with a
                                delay of 100ms between values
while(True):
    travel(path, speed, 0.1)
```

The ***travel*** function is the function that loops through the coordinates at the specified rate, creates the corresponding CAM message and publishes it to the internal broker:

```
def travel(street_list, speed_list, delay):
    j = 0
    for street in street_list:
        for key, coord in street.items():
            message = construct_message(coord[0], coord[1],
                                            speed_list[j])
            publish_message(message)
            time.sleep(delay)
        j += 1
        print("\n")
```

These generated CAM messages have the correct structure according to ETSI specifications and are published in the internal broker of the OBU in the topic *"vanetza/in/cam"*.

5.2 RSU CAM Processing and Intensity Calculation

In the following section, we explore the intricate process of Cooperative Awareness Message (CAM) processing and subsequent light intensity calculation by the Road-Side Units (RSUs). As critical components of our Smart Adaptive Lighting System, the successful implementation of these processes ensures that the system can dynamically adjust the light intensity based on real-time vehicular data.

To receive the CAM messages, we subscribe to the topic *"vanetza/out/-cam"* on the internal broker of the RSU. Then, every time the RSU receives a new CAM message, it makes the necessary computation and relays important information to the surrounding posts. Breaking down the computation process performed by the RSU upon receipt of a new CAM message, it becomes clear that this procedure can be divided into several distinct but interrelated stages:

1. **Range Verification:** The system first determines if the On-Board Unit (OBU) is within the defined communication range. If it's not, the corresponding Cooperative Awareness Message (CAM) is discarded. This method was chosen for its compatibility with scenarios involving multiple OBUs, as it circumvents the need to use the *"block MAC"* command on the containers.

2. **OBU Position Tracking:** The position of the OBU is logged in a dictionary, which holds the locations of all OBUs within range. If an OBU moves out of range, its entry is promptly removed from the dictionary.

3. **Proximity Assessment:** The system identifies the closest OBU to the Road-Side Unit (RSU), which becomes the focus for subsequent computations.

4. **RSU Light Intensity Calculation:** Utilizing the data provided by the CAM, the system calculates the appropriate light intensity for the RSU itself.

5. **Nearby Post Intensity Calculation:** The system extends its computations to encompass all lamp posts within a predefined radius of the RSU, calculating the optimal light intensity for each.

6. **Light Support Message (LSM) Generation and Publishing:** Finally, an LSM is generated based on the calculated light intensities. This message is then published to ensure seamless communication between the RSU and the surrounding lamp posts.

5.2.1 Range Verification & OBU Position Tracking

The RSU extracts the relevant information from the CAM message and calculates it's distance to the OBU:

```
longitude = message["longitude"]
latitude = message["latitude"]
speed = message["speed"]
obu_id = message["stationID"]
distance_between_car_and_post = round(distance((latitude,
                                longitude), (post.x, post.y)).
                                meters,2)
if(distance_between_car_and_post > RADIUS):
    #remove the OBU info from the dictionary
    OBUS[obu_id]["longitude"] = -1
    OBUS[obu_id]["latitude"] = -1
    OBUS[obu_id]["speed"] = -1
    #if no OBU is in range, alert other posts to remove this
                                   RSU from the ordering_rsus
    (...)
    publish_lsm(out_message)
    return
else:
    #Add the OBU to the dictionary
    OBUS[obu_id] = {"longitude": longitude, "latitude":
                                   latitude, "speed": speed}
```

When the CAM message specifies that the OBU falls within the effective range, the RSU updates the OBUs dictionary with this fresh data. Conversely, if the OBU strays outside of the designated range, the RSU deletes that OBU's corresponding information. In instances where no OBU remains within the RSU's radius, a Light Support Message (LSM) containing an intensity value of -1 is dispatched to the surrounding lamp posts. This negative value acts as a signal, notifying the network that the RSU is no longer processing CAM messages or determining light intensities.

5.2.2 Proximity Assessment

We developed a function that finds the closest OBU to the RSU, that is in range:

```
def get_closest_obu(OBUS, post_lat, post_long):
    closest_distance = 100000
    closest_obu = -1
    #loop through all the obus and find the closest one
    for key, value in OBUS.items():
        obu_lat = value['latitude']
        obu_long = value['longitude']
        #if the obu is not in range, skip it
        if obu_lat == -1 and obu_long == -1:
            continue

        #calculate the distance between the obu and the post
                                       using the distance
                                       function from geopy
        distance_between_car_and_post = round(distance((obu_lat
                                       , obu_long), (post_lat,
                                        post_long)).meters,2)

        #if the distance is smaller than the closest distance,
                                       update the closest
                                       distance and the
                                       closest obu
        if distance_between_car_and_post < closest_distance:
            closest_distance = distance_between_car_and_post
            closest_obu = key


    return closest_distance, closest_obu
```

This function returns the distance, in meters, to the closest OBU, and the corresponding ID.

5.2.3 RSU Light Intensity Calculation

Determining the appropriate light intensity for the RSU, based on the OBU's position and speed and a bias, necessitated the development of a custom function. This function first calculates the estimated arrival time, in seconds, of the OBU at the post's location. It then computes the intensity value, which exhibits an inverse proportionality to the arrival time and direct proportionality to the bias. The resulting intensity value is then mapped to fall within a predefined range from 20 to 100.

```
def calc_intensity(distance, speed, bias):
    #calculate the time it takes the OBU to get to the post in
                                seconds
    arrival_time = calc_interval(distance, speed)

    #calculate the luminosity
    #it is inversely proportional to the arrival time and
                                directly proportional to
                                the bias
    luminosidade = 1/arrival_time*100*bias

    #limit the luminosity to 100 and 20
    if luminosidade > 100:
        luminosidade = 100

    if luminosidade <20:
        luminosidade = 20

    #if the distance is less than 20 meters, the intensity is
                                always 100
    if distance < 20:
        return 100

    #return the luminosity rounded up
    return Math.ceil(luminosidade)
```

The value returned from the function above is saved to a global variable *MY_INTENSITY* to be later used in the construction of the LSM message.

5.2.4 Nearby Post Intensity Calculation

Following the computation of the RSU's intensity, our system extends its calculations to the surrounding smart lamp posts, ensuring the light intensities correspond to the nearest OBU to each respective post. This is achieved through the use of an external MQTT broker, deployed on our host machine and integrated with the Docker network. A script periodically publishes a data structure containing the lamp posts' information (latitude, longitude, status, and if it's an RSU) to a dedicated *'posts_info'* topic. Here's an illustrative example:

```
{
    '1': {
        'latitude': 40.636032,
        'longitude': -8.646632,
        'status': 'On',
        'rsu': true
    },
    '2': {
        'latitude': 40.635939,
        'longitude': -8.646695,
        'status': 'On',
        'rsu': false
    },
    (...)
}
```

Each RSU, being a subscriber to this topic on the broker, is kept informed about all posts and their respective geographical positions. Leveraging this data, the RSU identifies those posts within a certain radius of itself and computes their required light intensities, employing a calculation method similar to its own.

```
def get_times_to_arrival(in_range):
    global LAMPS, OBUS, RADIUS, POST_RANGE
    #for all the LAMPS, calculate the time to arrival of the
                                          closest obu
    times = {}
    for key, value in LAMPS.items():
        lat = value['latitude']
        lon = value['longitude']
        distance_val, obu_id = get_closest_obu(OBUS, lat, lon)
        difference_between_posts = round(distance((lat, lon), (
                                        post.x, post.y)).meters
                                        ,2)
        if difference_between_posts < POST_RANGE:
            if(in_range):
                time_to_arrival = round(calc_interval(
                                                distance_val,
                                                OBUS[obu_id]["
                                                speed"]),2)
                times[key] = time_to_arrival
            else:
                times[key] = -1

    return times
```

```
def get_intensities(times, bias):
    #for all the arrival times of the posts, calculate the
                                    corresponding intensity
    intensities = {}
    for key, value in times.items():
        #intensity_on_time is the same as calc_intensity
                                        described before,
                                        taking time as an
                                        argument instead of
                                        distance and speed
        temp = intensity_on_time(value,  bias)
        if (temp > 20):
            intensities[key] = temp
        elif (value == -1):
            #if the obu is not in range, the intensity is -1
            intensities[key] = -1

    return intensities
```

Both these functions provide the necessary information for the RSU to relay to the other posts, where the dictionary *intensities* maps the light intensity value with the corresponding post ID. Now this information just needs to be published to the external broker in the *'all/lsm'* topic.

5.2.5 Light Support Message (LSM) Generation and Publishing

As we stated before, the external MQTT broker serves as a pivotal intermediary for message handling between the cars and lampposts. It facilitates the seamless exchange of messages, including the LSMs, which are specialized messages designed to convey intensity information. The broker now acts as a centralized hub for receiving, processing, and distributing LSMs between the lampposts.

These messages are generated by the RSUs themselves and contain essential details such as the lamppost's unique identifier, geographical coordinates, it's own intensity level and a dictionary containing the destination stations and their corresponding intensity (the result of the *get_intensities* function). Additionally, LSMs may include information about associated OBU closest to itself. These attributes enable the lampposts to effectively evaluate the intensity levels and make informed decisions based on the received LSMs.

As an example, consider the following LSM message:

```
{
    "station_id": 9,
    "station_latitude": 40.636032,
    "station_longitude": -8.646632,
    "dest_stations": {"11":48, "9":70, "2":100},
    "intensity": 70,
    "obu_latitude": 40.638049,
    "obu_longitude": -8.649238,
    "obu_id": 1,
    "timestamp": "2023-05-17 12:24:32:0"
}
```

To get a full overview of the overall computation process integrated into the RSUs:

```
#out of all the obus, get the closest distance
closest_distance, closest_obu = get_closest_obu(OBUS, post.x,
                                 post.y)
#calculate the intensity of the RSU
iluminacao = calc_intensity(closest_distance,speed, BIAS)
MY_INTENSITY = iluminacao

#calculate the intensities for the other posts in a given range
                                 and send the message
times = get_times_to_arrival(True)
intensities = get_intensities(times, BIAS)
out_message = construct_message([2], MY_INTENSITY, intensities,
                                 latitude, longitude, obu_id)
publish_lsm(out_message)
```

5.3 API and Data Gathering

The ”API and Data Gathering” section focuses on the deployment of an API to collate the LSM messages. The API, subscribing to the topic ’all/lsm’ on the external broker, processes the messages and offers the data through designated endpoints. Our solution makes use of this API for three main purposes: LSM message reception and processing, simulation of Smart Lamp Post behavior and real-time updates on the location and IDs of the OBUs.

The behavior of the Smart Lamp Posts is captured in a dictionary called *POST_STATUS*, which provides a snapshot of each lamp post (RSU or non-RSU) including its intensity, target posts with their respective intensities, an identifier for the RSU issuing the orders, and geographical coordinates.

Here’s a simplified example of the *POST_STATUS* dictionary:

```json
{
  "15": {
    "intensity": 100,
    "target_posts": {
      "9": 63,
      "18": 64,
    },
    "ordering_rsu_id": 15,
    "in_range": true,
    "lat": 40.637318,
    "lon": -8.649641,
    "status": "On",
    "rsu": true
  },
   "16": {
    "intensity": 36,
    "target_posts": {
      "13": 50,
      "18": 30,
    },
    "ordering_rsu_id": 16,
    "in_range": true,
    "lat": 40.638006,
    "lon": -8.649255,
    "status": "On",
    "rsu": true
  },
  "18": {
    "lat": 40.637462,
    "lon": -8.648595,
    "status": "On",
    "rsu": false,
    "intensity": 64,
    "in_range": false,
    "ordering_rsu_id": 15,
    "target_posts": [],
    }
  ...
}
```

This dictionary has two main applications: Firstly, it's used to continuously update a log file (log.txt), providing an ongoing record of system activity. Secondly, it serves data for the *'/api/v1/rsu_data'* API endpoint:

```
@app.route('/api/v1/rsu_data', methods=['GET'])
def get_rsu():
    global POSTS_status
    return POSTS_status, 200
```

Given that non-RSU posts could potentially receive multiple intensity orders from various RSUs concurrently, we implemented another dictionary called *ORDERING_INTENSITIES*. This dictionary maps each post to the IDs of the RSUs ordering it and the corresponding intensity values.

```
{
    "1":{
        "2":30,
        "4": 23,
        "12": 89,
    },
    "2":{
        "6":34,
        "8": 43,
        "22": 54,
    },
...
}
```

This dictionary is filled and updated with the information coming in from the LSM messages, specially in the *"dest_stations"* parameter:

```
intensity = message["intensity"]
dest_stations = message["dest_stations"]
POSTS_status[id]['intensity'] = intensity
POSTS_status[id]['target_posts'] = dest_stations

#for each post, adds the intensity that it receives from the
                         rsu (id)

for key, value in ORDERING_INTENSITIES.items():
    if key in dest_stations:
        intensity = dest_stations[key]
        if intensity == -1:
            if id in ORDERING_INTENSITIES[key]: # Only delete
                                        if the key exists
                del ORDERING_INTENSITIES[key][id]
        else:
            ORDERING_INTENSITIES[key][id] = dest_stations[key]
```

To determine the intensity to apply, each post simply selects the maximum intensity value from its list of ordering RSUs in this dictionary. The 'intensity' attribute in the post's entry in *POST_STATUS* is then updated accordingly.

```
#for each post, gets the max intensity that it receives from
                                the rsus
if POSTS_status != {}:
    #Loop through all the posts and get the max intensity for
                                    each post and the rsu that
                                    sends it
    for key, value in ORDERING_INTENSITIES.items():
        #robustness
        if key not in POSTS_status:
            POSTS_status[str(key)] = {}

        #get the max intensity and the rsu that sends it
        max = 0
        id_max = -1
        for key2, value2 in value.items():
            if value2 > max:
                max = value2
                id_max = key2
        if max == 0:
            max = 20

        #update the POSTS_status
        POSTS_status[key]['intensity'] = max
        POSTS_status[key]['ordering_rsu_id'] = int(id_max)
        if POSTS_status[key]['in_range'] == True:
            POSTS_status[key]['ordering_rsu_id'] = int(key)
```

Additionally, when an RSU loses connection with an OBU and dispatches an LSM message with an intensity of -1 to its related posts, that RSU is removed from the *ORDERING_INTENSITIES* dictionary. This indicates that the RSU is no longer issuing intensity orders.

The API also plays a vital role in providing real-time updates on the location of the OBUs. This information is made accessible via the '/api/v1/obu' endpoint, which serves the OBU data from a globally accessible dictionary, OBUS:

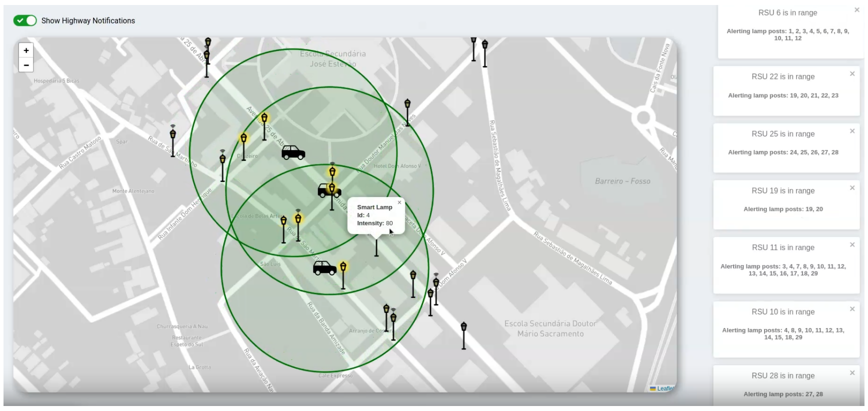
```
@app.route('/api/v1/obu', methods=['GET'])
def get_obu():
    global OBUS
    return OBUS, 200
```

The OBUS dictionary is populated and updated based on the CAM messages being received by one of the RSUs. Specifically, the API subscribes to the topic "vanetza/out/cam" of one of the RSUs, and its incoming messages serve to update the OBUs' current state. This approach significantly streamlines our operational and simulation processes, by providing a singular, real-time source of truth for the OBUs' status.

5.4 Front-End Application

In this section, we'll explore the Front-End Application that provides a real-time visualization of our simulation. This application was built using React.js and incorporates an OpenStreetMap to represent the real-world environment where our OBUs and RSUs operate, integrated using the React-Leaflet library. This aids in the effective visualization of the RSUs, OBUs, and Smart Lamp Posts in real time.

The map is populated with icons representing OBUs and both RSUs and Smart Lamp Posts. These icons are constantly updated to accurately portray the simulation in real time. The RSU and Smart Lamp Post icons come equipped with a dynamic yellow glow that reflects the current light intensity. A clickable popup is also available on these icons, providing more detailed information about the post, such as its ID, current light intensity, and whether it's within the range of an OBU (if it's an RSU).



Smart Lamp
Id: 4
Intensity: 80

RSU 6 is in range
Alerting lamp posts: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12

RSU 22 is in range
Alerting lamp posts: 19, 20, 21, 22, 23

RSU 25 is in range
Alerting lamp posts: 24, 25, 26, 27, 28

RSU 19 is in range
Alerting lamp posts: 19, 20

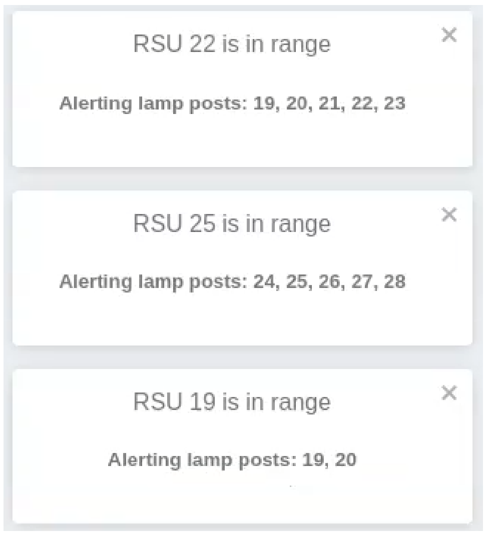
RSU 11 is in range
Alerting lamp posts: 3, 4, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 29

RSU 10 is in range
Alerting lamp posts: 4, 8, 9, 10, 11, 12, 13, 14, 15, 18, 29

RSU 28 is in range
Alerting lamp posts: 27, 28

Figure 5.1: Full Web Platform

Additionally, our Front-End Application incorporates a minimal in-app notification system to provide further insights into the system's real-time operation. This system delivers notifications when an OBU enters the range of a certain RSU, providing explicit information about which Smart Lamp Posts that specific RSU is issuing commands to.


RSU 22 is in range

Alerting lamp posts: 19, 20, 21, 22, 23

RSU 25 is in range

Alerting lamp posts: 24, 25, 26, 27, 28

RSU 19 is in range

Alerting lamp posts: 19, 20

Figure 5.2: Notification System

These live updates offer an additional layer of context to the user, aiding in the understanding of how RSUs interact with OBUs and manage surrounding Smart Lamp Posts.

The purpose of this platform is to offer an intuitive, visual representation of the simulation, offering users an accessible way to understand and interact with the system dynamics. The front-end application achieves this by fetching data in real time from the previously mentioned API endpoints: *'/api/v1/rsu_data'* for the RSUs and Lamp posts information and *'/api/v1/obu'* for the OBUs' locations. This real-time data communication between the API and the front-end ensures an accurate, up-to-the-second representation of the simulation's current state.

5.5 Challenges and Solutions

During the course of this project, we encountered several obstacles that challenged our initial plans and demanded innovative solutions. These challenges not only tested our technical knowledge and problem-solving skills but also prompted us to seek out more efficient, robust, and scalable solutions. In this section, we will outline the major challenges that we faced during the implementation of our system, along with the strategies and solutions that we devised to overcome them. From fine-tuning the calculation of light intensity to handling multiple On-Board Units (OBUs) and Roadside Units (RSUs), these experiences shaped the evolution of our system and played a crucial role in achieving our objectives.

1. **Light Intensity Calculation:** The initial challenge was to devise an effective formula that can map speed and distance (or arrival time)

to an appropriate light intensity level while factoring in a bias. The solution was the creation of a mathematical equation that inversely correlates the arrival time with the light intensity and includes a bias multiplier to account for varying rates of light intensity among different posts under the same conditions.

2. **Optimization of Resources:** A key challenge was to minimize the usage of system resources and network bandwidth, especially when dealing with larger messages. The solution was to limit the computation of light intensity values to nearby posts only, defined by a RADIUS parameter during program execution. This dynamic adjustment method was particularly useful in scenarios with fewer RSUs, where a larger radius was beneficial to notify all the posts effectively.

3. **Multiple OBUs Interaction:** Upon introducing multiple OBUs into the system, the lights started to flicker due to the reception of different intensity orders from various RSUs. Initially, this was addressed by selecting the highest value. However, this led to the lights remaining continuously on, even in the absence of OBUs. This was remedied by implementing a feature where an RSU sends a LSM message with -1 intensity to the surrounding posts upon going out of OBU range. This triggered the removal of this RSU from the ordering list and recalculation of the light intensity based on remaining RSUs.

4. **Script Unification:** We needed a versatile and scalable solution to handle an arbitrary number of OBUs and RSUs in the system. Therefore, we developed a single Python script for both OBUs and RSUs. For the OBU program, parameters include: id, container IP, and simulation path (1-5). For the RSU program, parameters include: ID, latitude, longitude, BIAS, IP, and destination posts range. This approach allowed us to execute the same program as many times as necessary, tailored to the specific parameters of each OBU or RSU in the system.

5.6 Testing

In order to validate the functionality and robustness of our system, we designed two distinct testing scenarios that simulated real-world conditions. Our first scenario involved a highway setting, where two vehicles navigated through a zone in opposite directions. This scenario was created to examine the responsiveness of the lamp posts as the vehicles traversed the zone at high speeds. Our second scenario was modeled after an urban environment, involving three vehicles following distinct closed paths. This scenario allowed us to observe the system's performance in a dynamic setting, where multiple vehicles interact with the Road Side Units (RSUs) in various patterns, both simultaneously and at separate times.

These testing scenarios provided valuable insights into the system's behavior under different circumstances. After thorough examination and analysis, we concluded that our system performed admirably in both scenarios. It exhibited robust responsiveness and accurate light intensity modulation, demonstrating its potential for real-world applications. These results serve as a testament to the effectiveness of our implementation and the success of our project

Chapter 6

Flow of the Demo

Our demonstration, realized through a dynamic React application, displays two distinct scenarios to showcase the functionality of the Smart Adaptive Lighting System - an urban scenario and a highway scenario. Both settings feature vehicles (OBUs) operating in different conditions and a variety of lamp posts (RSUs and non-RSUs), together illustrating the system's adaptability and efficiency in varying contexts.

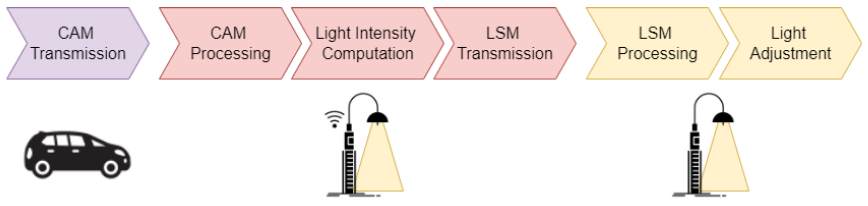


Figure 6.1: Message Transmission Timeline

In the urban scenario, we simulate three OBUs navigating predetermined closed paths within a cityscape. Their routes are designed so that the vehicles cross paths in both similar and opposite directions, allowing us to showcase the system's ability to manage multiple inputs simultaneously. As these OBUs traverse their routes, they come within the communication range of several smart lamp posts, both RSUs and non-RSUs. Viewers of the demo can observe notifications indicating when the RSUs come within the transmission range of OBUs' CAM messages. A captivating feature of this scenario is the real-time visualization of the dynamic lighting system. As the vehicles approach the smart lamp posts, the intensity of the light grows, with neighboring lamp posts also adjusting their intensity in accordance with their relative location to the vehicle. The visual representation of this intelligent response provides a clear and intuitive understanding of the system's functionality.

The highway scenario provides another perspective on the system's capabilities. Here, two OBUs traverse a lamp post-rich zone in opposite directions. Similar to the urban scenario, viewers can observe the changing intensity of the lamp posts as the vehicles approach, pass by, and move away. This scenario is particularly effective in illustrating the adaptability of the lighting system at higher speeds and in environments with a dense distribution of lamp posts.

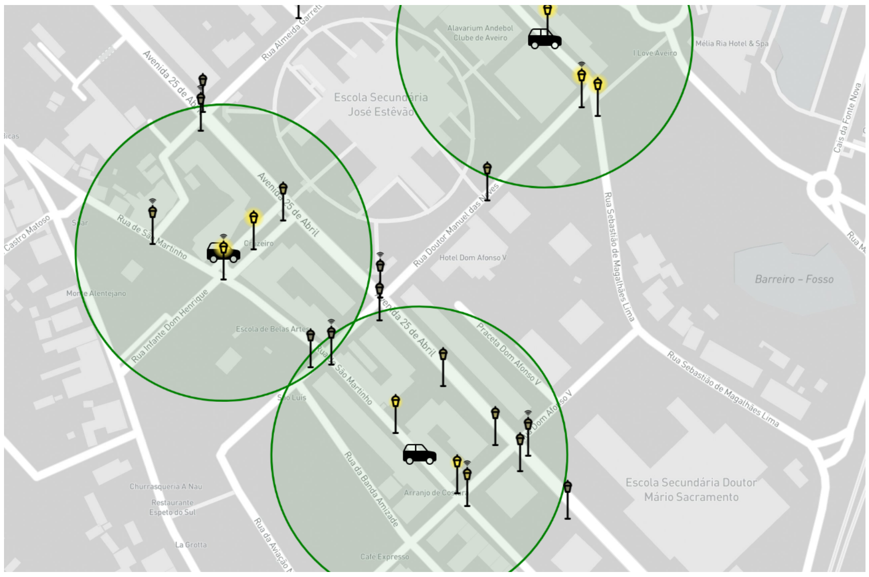

Escola Secundária José Estêvão
Avenida 25 de Abril
Rua de São Martinho
Rua Infante Dom Henrique
Rua Doutor Manuel das Neves
Rua Sebastião de Magalhães Lima
Praceta Dom Afonso V
Rua da Banda Amizade
Hotel Dom Afonso V
Barreiro – Fosso
Escola Secundária Doutor Mário Sacramento
Alavarium Andebol Clube de Aveiro
Mélia Ria Hotel & Spa

Figure 6.2: Map visualization with OBUs, RSUs and Smart Lamp Posts

These two scenarios, with interactive icons representing OBUs and lamp posts, provide a comprehensive demonstration of the Smart Adaptive Lighting System's capabilities. By observing these scenarios, viewers can appreciate how this system can enhance both energy efficiency and safety on roads in real-world driving conditions.

Chapter 7

Results

The Smart Adaptive Lighting System has been successfully developed and implemented, achieving all the set project objectives. Notably, the system has shown remarkable robustness and adaptability, effectively handling complex scenarios with multiple OBUs in a given zone. The flawless operation under these demanding conditions validates the robustness of the system and its suitability for deployment in real-world contexts.

Our real-time visualization platform is one of the major highlights of the project. Providing real-time, intuitive representations of vehicle movements and dynamic changes in lighting intensity, the platform significantly aids user understanding of the system's functionality. Also, the correct dynamic calculations of the light intensity values by the RSUs are at the core of this project.

Energy efficiency, one of the cornerstone goals of this project, has been addressed through the intelligent adaptive lighting. While exact calculations of potential energy savings require field testing, we anticipate substantial electricity savings given the system's ability to dynamically adjust lighting based on vehicular presence.

Overall, the Smart Adaptive Lighting System has demonstrated strong potential to contribute positively to smart city initiatives and road safety, fully achieving its intended goals at this stage of development.

Chapter 8

Future Work

In looking forward, there are ample opportunities to expand and refine the capabilities of the Smart Adaptive Lighting System. A promising area for further development is the integration of RADAR sensors in some RSUs. This would enable the detection of vehicles that lack OBUs, such as older car models or bicycles, thereby extending the system's benefits to all road users, not just those in equipped vehicles.

Furthermore, we envision the use of AI-enabled object detection cameras in RSUs (based on YOLO), capable of identifying and tracking non-motorized road users, such as pedestrians and cyclists. This technology provides accurate location tracking for these road users, triggering the adaptive lighting system and thereby ensuring optimal visibility and enhanced safety for everyone sharing the road.

Expanding the system's capability to interact with a wider range of road users will bring us closer to a truly inclusive smart city infrastructure.

Chapter 9

Link to the Repository

The link to the project's repository with all the files and configurations is the following: `https://github.com/JoaoantonioViegas/RSA_project`

Chapter 10

Link to the Video of the Demo

The link to the project's demo is the following: `https://drive.google.com/file/d/17vO4MNCiZPNG-qULRueaSjVUfjc3zF6h/view?usp=sharing`