Walk and Seat

Universidade de Aveiro

Manuel Couto, Bernardo Falé

Walk and Seat

DETI - Redes e Sistemas Autónomos

Universidade de Aveiro

Manuel Couto, Bernardo Falé
(93285) manuelcouto10@ua.pt, (93331) mbfale@ua.pt

14/06/2023

Contents

1 Objectives of the work **1**

2 Architecture **2**

2.1 Implementation 3

2.1.1 Roadside Unit (RSU) 3

2.1.2 Vulnerable Road User (VRU) 4

2.1.3 Vulnerable Road User Awareness Message (VAM) 4

2.1.4 InterPlanetary File System (IPFS) 4

2.1.5 Flask Web Server 4

2.1.6 CID Tracker DataBase 4

2.2 Architecture Illustration 4

3 Flow of the demo **7**

4 Results **8**

5 GitHub Repository and Demo **10**

Chapter 1

Objectives of the work

The objective of our work was to explore a different approach of VANETZA, which deviates from the traditional vehicle-centric focus of VANET(Vehicular Ad Hoc Network). Instead, we aimed to apply VANETZA principles to individuals, creating a network based on persons. Our aim was to provide clients in the vicinity of the restaurants with access to relevant content, such as the number of available tables and menus.

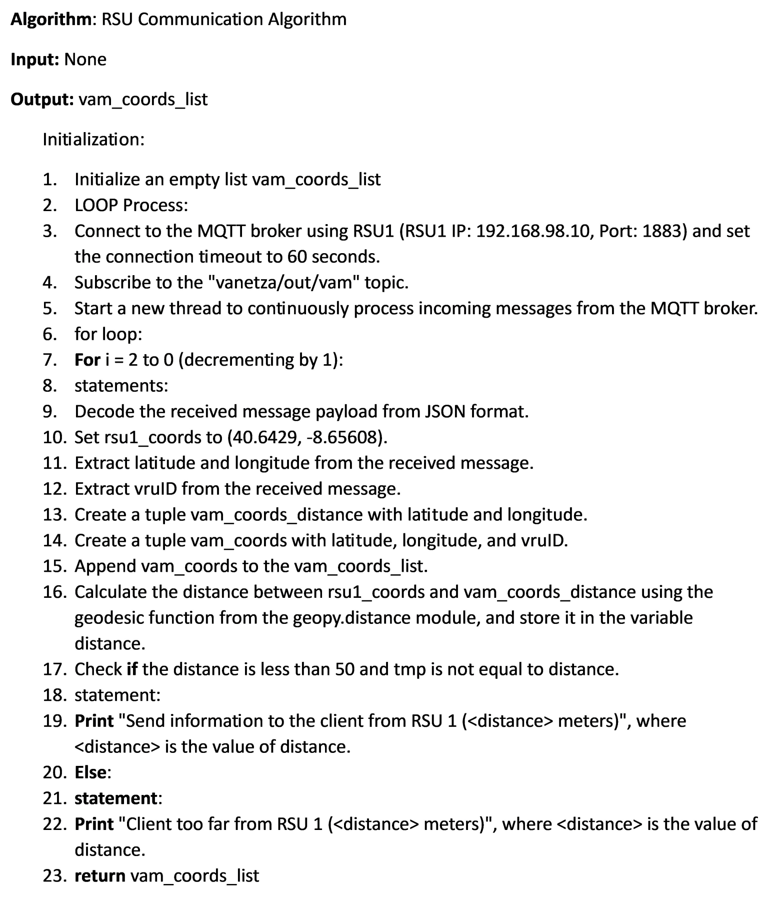
To achieve this, we proposed implementing a cluster of restaurants that would enable distributed and rapid dissemination of information among clients. In this approach, each person would function as a VRU and serve as a node within the IPFS network. Similarly, each restaurant would act as a RSU and also function as an IPFS node.

We intended to facilitate efficient communication between clients and restaurants, allowing clients to easily access real-time information about the availability of tables and menu options. The use of IPFS as the underlying network protocol would enable decentralized, distributed and robust information sharing within the VANETZA network and allow offline availability and authenticity of information.

Overall, our objective was to explore the feasibility and potential benefits of implementing a VANETZA-based system in the context of restaurant communication and content dissemination.

Chapter 2

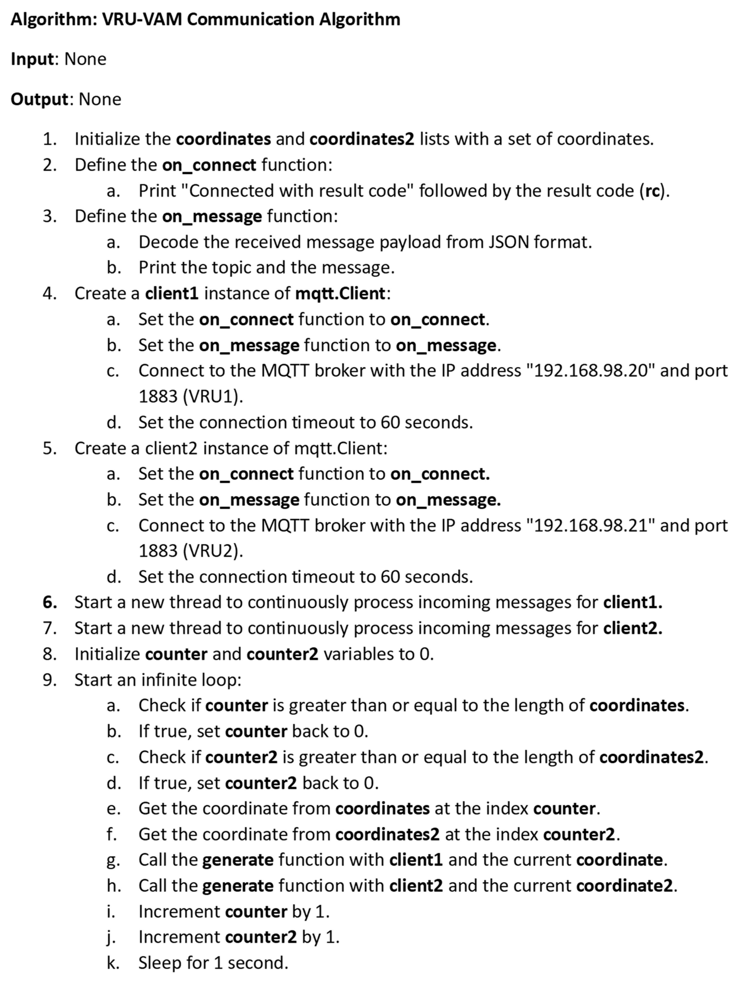
Architecture



```
Algorithm: RSU Communication Algorithm
Input: None
Output: vam_coords_list
    Initialization:
    1.  Initialize an empty list vam_coords_list
    2.  LOOP Process:
    3.  Connect to the MQTT broker using RSU1 (RSU1 IP: 192.168.98.10, Port: 1883) and set
        the connection timeout to 60 seconds.
    4.  Subscribe to the "vanetza/out/vam" topic.
    5.  Start a new thread to continuously process incoming messages from the MQTT broker.
    6.  for loop:
    7.  For i = 2 to 0 (decrementing by 1):
    8.  statements:
    9.  Decode the received message payload from JSON format.
    10. Set rsu1_coords to (40.6429, -8.65608).
    11. Extract latitude and longitude from the received message.
    12. Extract vruID from the received message.
    13. Create a tuple vam_coords_distance with latitude and longitude.
    14. Create a tuple vam_coords with latitude, longitude, and vruID.
    15. Append vam_coords to the vam_coords_list.
    16. Calculate the distance between rsu1_coords and vam_coords_distance using the
        geodesic function from the geopy.distance module, and store it in the variable
        distance.
    17. Check if the distance is less than 50 and tmp is not equal to distance.
    18. statement:
    19. Print "Send information to the client from RSU 1 (<distance> meters)", where
        <distance> is the value of distance.
    20. Else:
    21. statement:
    22. Print "Client too far from RSU 1 (<distance> meters)", where <distance> is the value of
        distance.
    23. return vam_coords_list
```

Figure 2.1: RSU Algorithm

The Algorithm for RSU2 is identical at 2.1



```
Algorithm: VRU-VAM Communication Algorithm

Input: None

Output: None

    1. Initialize the coordinates and coordinates2 lists with a set of coordinates.
    2. Define the on_connect function:
        a. Print "Connected with result code" followed by the result code (rc).
    3. Define the on_message function:
        a. Decode the received message payload from JSON format.
        b. Print the topic and the message.
    4. Create a client1 instance of mqtt.Client:
        a. Set the on_connect function to on_connect.
        b. Set the on_message function to on_message.
        c. Connect to the MQTT broker with the IP address "192.168.98.20" and port
           1883 (VRU1).
        d. Set the connection timeout to 60 seconds.
    5. Create a client2 instance of mqtt.Client:
        a. Set the on_connect function to on_connect.
        b. Set the on_message function to on_message.
        c. Connect to the MQTT broker with the IP address "192.168.98.21" and port
           1883 (VRU2).
        d. Set the connection timeout to 60 seconds.
    6. Start a new thread to continuously process incoming messages for client1.
    7. Start a new thread to continuously process incoming messages for client2.
    8. Initialize counter and counter2 variables to 0.
    9. Start an infinite loop:
        a. Check if counter is greater than or equal to the length of coordinates.
        b. If true, set counter back to 0.
        c. Check if counter2 is greater than or equal to the length of coordinates2.
        d. If true, set counter2 back to 0.
        e. Get the coordinate from coordinates at the index counter.
        f. Get the coordinate from coordinates2 at the index counter2.
        g. Call the generate function with client1 and the current coordinate.
        h. Call the generate function with client2 and the current coordinate2.
        i. Increment counter by 1.
        j. Increment counter2 by 1.
        k. Sleep for 1 second.
```

Figure 2.2: VRU Algorithm

2.2The generate function, take a client and a coordinate as input, generate a JSON message using the coordinate information, and publish it to the corresponding MQTT topic ("vanetza/in/vam").

2.1 Implementation

2.1.1 RSU

We decided to use 2 RSUs in our simulation, each RSU represents one restaurant, the RSUs were simulated using Message Queuing Telemetry Transport (MQTT) brokers. These RSU's subscribed to the VAMs topics, enabling them to receive and process notifications.

2.1.2 VRU

We opted to simulate 2 persons, so we simulate 2 VRUs too, the VRUs were simulated using MQTT brokers. These brokers published VAM messages.

2.1.3 VAM

VAMs, meaning Vulnerable Road User Awareness Message, are the type of messages that we are using in our network.

2.1.4 IPFS

To spread information in a decentralized and distributed way, and to ensure offline availability and authenticity of information we choose to use IPFS, to improve even more we implemented a ipfs cluster improving the spread of information across the network.

2.1.5 Flask Web Server

In order to enhance the user experience of our project, we have created a user-friendly Flask web application. This application effectively showcases the simulated environment and provides real-time updates on all ongoing activities. To access this feature, users simply need to follow a few steps: running the VANETZA docker, executing the script responsible for simulating the VRUs and launching the Flask web application. Additionally, the web application conveniently includes the Roadside Unit (RSU) scripts, which automatically initiate upon startup. This streamlined process ensures a seamless and efficient experience for our users.

2.1.6 CID Tracker DataBase

A content identifier, or CID, is a label used to point to material in IPFS. It doesn't indicate where the content is stored, but it forms a kind of address based on the content itself. CIDs are short, regardless of the size of their underlying content. CIDs are based on the content's cryptographic hash so if the content changes the CID will not match, improving the security of our project, we use a database to address each CID with a restaurant name, basically working as a tracker.

2.2 Architecture Illustration

The Figure 2.4 illustrates the architecture of our system, which incorporates IPFS technology and encompasses VRUs and RSUs deployed within the VANETZA

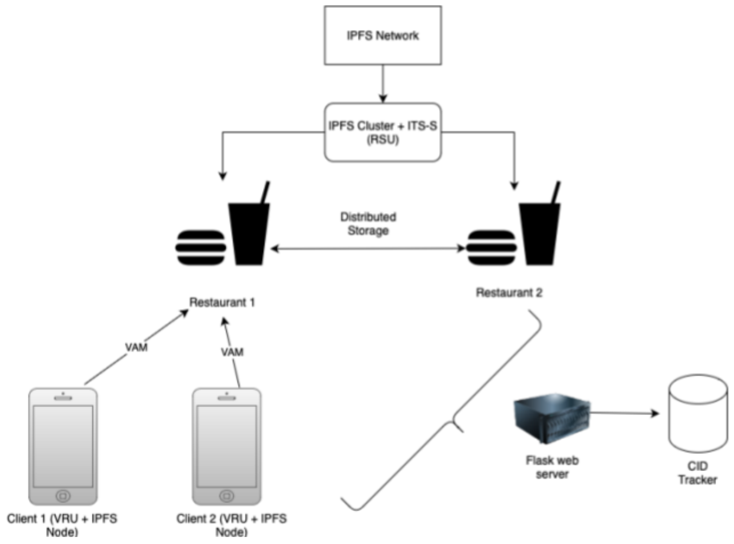


Figure 2.3: Implementation Diagram

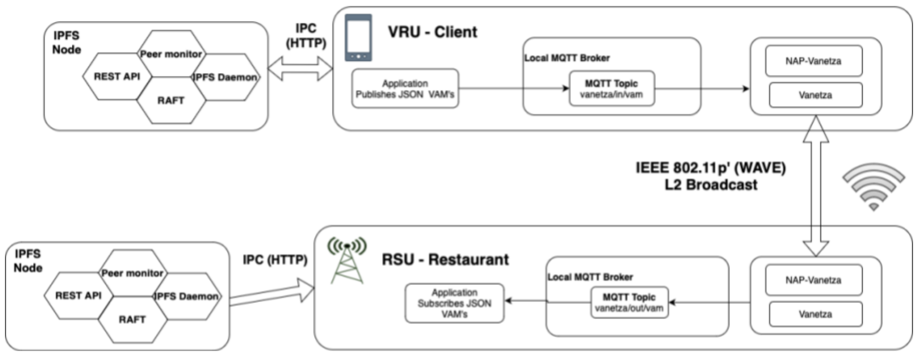


Figure 2.4: System Architecture

Docker. Both VRUs and RSUs are simulated using an MQTT broker, with the VRU publishing messages to the broker and the RSU subscribing to them. These messages are generated by a script that publishes VAMs sent from the clients. When a Restaurant/RSU, after processing a VAM message detects a VRU within a proximity of less than 50 meters, it initiates the transmission of data. The distance between the RSU and the VRUs is calculated in each RSU that subscribes to the vanetza/out/vam topic. Python scripts are utilized by both RSUs.

Chapter 3

Flow of the demo

The first step in the process is the deployment of the IPFS cluster, VANETZA containers, and the web application. This involves setting up the necessary infrastructure to support communication and data storage between these components. Once the deployment is complete, the system starts generating messages through the MQTT broker. These messages facilitate communication between the different components of the system.

During the demonstration, the VRUs follow a predefined path of coordinates within the Aveiro area. As the VRUs move, their proximity to a certain area is determined by a circle radius. When a VRU enters this circle area, it indicates that they are within the range of content distribution. If a VRU is within the green area, the information relevant to that location is displayed in cards on the web page. If a VRU is within the circle area and the information is being distributed, it will be displayed on the web page. Even if a VRU moves out of the circle area, they can still access the previously received information, as it becomes "theirs." This ensures that VRUs have access to relevant information even when they are no longer within the distribution range. Although not fully implemented, the system allows clients to book a table and wait for the restaurant to manually update the information. This feature enables clients to interact with the system and make reservations for tables. The restaurant can then update table availability and other relevant information as needed.

Chapter 4

Results

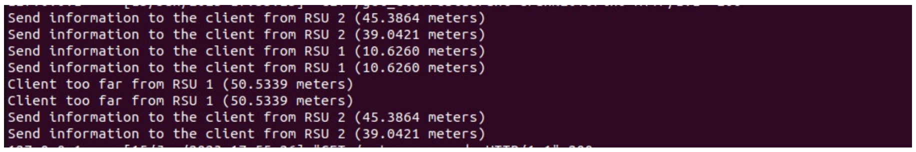
The deployment phase of the project was executed successfully, resulting in the successful setup and configuration of the IPFS cluster (as seen in Fig. 4.3), VANETZA containers, and the web application. This section presents the key outcomes and findings from the deployment process, highlighting the setup process and the challenges overcome.

The setup process for the IPFS cluster involved configuring multiple IPFS nodes and establishing their collaborative functionality. Each node was assigned specific roles and responsibilities to ensure efficient content storage and retrieval. Through meticulous coordination and synchronization, a cohesive IPFS cluster was formed, enabling effective data distribution and access.

Simultaneously, the VANETZA containers were deployed to facilitate communication between clients and the IPFS cluster. These containers were carefully configured to handle data transmission and reception, serving as the intermediary interface within the system.

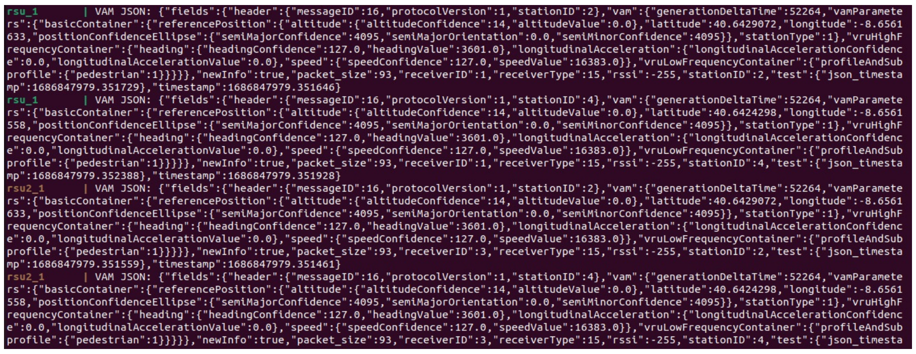
During the deployment process, various challenges were encountered, demanding effective problem-solving and collaboration. One notable challenge was establishing efficient communication and synchronization between the IPFS cluster, VANETZA containers, and the web application. Despite the challenges faced, the deployment phase concluded with successful outcomes. The IPFS cluster, VANETZA containers, and the web application were configured and synchronized effectively, facilitating seamless communication and content distribution within the system.

During the simulation, the behavior of Vulnerable Road Users (VRUs) was closely monitored to analyze their interaction with the system and the effectiveness of content distribution. 4.1 presents an brief window of VRU behavior. The system employed a circle area to determine the proximity of VRUs to the content distribution zone. VRUs entering this circle area were considered within the range of content distribution. We made sure that both VRUs entered the circle area, indicating successful coverage of the target audience.



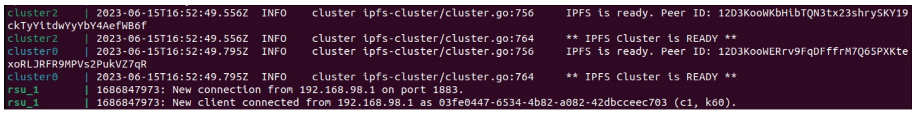
```
Send information to the client from RSU 2 (45.3864 meters)
Send information to the client from RSU 2 (39.0421 meters)
Send information to the client from RSU 1 (10.6260 meters)
Send information to the client from RSU 1 (10.6260 meters)
Client too far from RSU 1 (50.5339 meters)
Client too far from RSU 1 (50.5339 meters)
Send information to the client from RSU 2 (45.3864 meters)
Send information to the client from RSU 2 (39.0421 meters)
```

Figure 4.1: RSU distance prints



```
rsu_1       | VAM JSON: {"fields":{"header":{"messageID":16,"protocolVersion":1,"stationID":2},"vam":{"generationDeltaTime":52264,"vamParamete
rs":{"basicContainer":{"referencePosition":{"altitude":{"altitudeConfidence":14,"altitudeValue":0.0},"latitude":40.6429072,"longitude":-8.6561
633,"positionConfidenceEllipse":{"semiMajorConfidence":4095,"semiMajorOrientation":0.0,"semiMinorConfidence":4095}},"stationType":1},"vruHighF
requencyContainer":{"heading":{"headingConfidence":127.0,"headingValue":3601.0},"longitudinalAcceleration":{"longitudinalAccelerationConfidenc
e":0.0,"longitudinalAccelerationValue":0.0},"speed":{"speedConfidence":127.0,"speedValue":16383.0}},"vruLowFrequencyContainer":{"profileAndSub
profile":{"pedestrian":1}}}}},"newInfo":true,"packet_size":93,"receiverID":1,"receiverType":15,"rssi":-255,"stationID":2,"test":{"json_timesta
mp":1686847979.351729},"timestamp":1686847979.351646}
rsu_1       | VAM JSON: {"fields":{"header":{"messageID":16,"protocolVersion":1,"stationID":4},"vam":{"generationDeltaTime":52264,"vamParamete
rs":{"basicContainer":{"referencePosition":{"altitude":{"altitudeConfidence":14,"altitudeValue":0.0},"latitude":40.6424298,"longitude":-8.6561
558,"positionConfidenceEllipse":{"semiMajorConfidence":4095,"semiMajorOrientation":0.0,"semiMinorConfidence":4095}},"stationType":1},"vruHighF
requencyContainer":{"heading":{"headingConfidence":127.0,"headingValue":3601.0},"longitudinalAcceleration":{"longitudinalAccelerationConfidenc
e":0.0,"longitudinalAccelerationValue":0.0},"speed":{"speedConfidence":127.0,"speedValue":16383.0}},"vruLowFrequencyContainer":{"profileAndSub
profile":{"pedestrian":1}}}}},"newInfo":true,"packet_size":93,"receiverID":1,"receiverType":15,"rssi":-255,"stationID":4,"test":{"json_timesta
mp":1686847979.352388},"timestamp":1686847979.351928}
rsu2_1      | VAM JSON: {"fields":{"header":{"messageID":16,"protocolVersion":1,"stationID":2},"vam":{"generationDeltaTime":52264,"vamParamete
rs":{"basicContainer":{"referencePosition":{"altitude":{"altitudeConfidence":14,"altitudeValue":0.0},"latitude":40.6429072,"longitude":-8.6561
633,"positionConfidenceEllipse":{"semiMajorConfidence":4095,"semiMajorOrientation":0.0,"semiMinorConfidence":4095}},"stationType":1},"vruHighF
requencyContainer":{"heading":{"headingConfidence":127.0,"headingValue":3601.0},"longitudinalAcceleration":{"longitudinalAccelerationConfidenc
e":0.0,"longitudinalAccelerationValue":0.0},"speed":{"speedConfidence":127.0,"speedValue":16383.0}},"vruLowFrequencyContainer":{"profileAndSub
profile":{"pedestrian":1}}}}},"newInfo":true,"packet_size":93,"receiverID":3,"receiverType":15,"rssi":-255,"stationID":2,"test":{"json_timesta
mp":1686847979.351559},"timestamp":1686847979.351461}
rsu2_1      | VAM JSON: {"fields":{"header":{"messageID":16,"protocolVersion":1,"stationID":4},"vam":{"generationDeltaTime":52264,"vamParamete
rs":{"basicContainer":{"referencePosition":{"altitude":{"altitudeConfidence":14,"altitudeValue":0.0},"latitude":40.6424298,"longitude":-8.6561
558,"positionConfidenceEllipse":{"semiMajorConfidence":4095,"semiMajorOrientation":0.0,"semiMinorConfidence":4095}},"stationType":1},"vruHighF
requencyContainer":{"heading":{"headingConfidence":127.0,"headingValue":3601.0},"longitudinalAcceleration":{"longitudinalAccelerationConfidenc
e":0.0,"longitudinalAccelerationValue":0.0},"speed":{"speedConfidence":127.0,"speedValue":16383.0}},"vruLowFrequencyContainer":{"profileAndSub
profile":{"pedestrian":1}}}}},"newInfo":true,"packet_size":93,"receiverID":3,"receiverType":15,"rssi":-255,"stationID":4,"test":{"json_timesta
```

Figure 4.2: VAM Messages



```
cluster2    | 2023-06-15T16:52:49.556Z  INFO    cluster ipfs-cluster/cluster.go:756     IPFS is ready. Peer ID: 12D3KooWKbHibTQN3tx23shrySKY19
ckTyYitdwYyYbY4AefWB6f
cluster2    | 2023-06-15T16:52:49.556Z  INFO    cluster ipfs-cluster/cluster.go:764     ** IPFS Cluster is READY **
cluster0    | 2023-06-15T16:52:49.795Z  INFO    cluster ipfs-cluster/cluster.go:756     IPFS is ready. Peer ID: 12D3KooWERrv9FqDFffrM7Q65PXKte
xoRLJRFR9MPVs2PukVZ7qR
cluster0    | 2023-06-15T16:52:49.795Z  INFO    cluster ipfs-cluster/cluster.go:764     ** IPFS Cluster is READY **
rsu_1       | 1686847973: New connection from 192.168.98.1 on port 1883.
rsu_1       | 1686847973: New client connected from 192.168.98.1 as 03fe0447-6534-4b82-a082-42dbcceec703 (c1, k60).
```

Figure 4.3: Cluster Ready

Chapter 5

GitHub Repository and Demo

All the work developed is available in https://github.com/coutooo/RSA-PROJECT
and the demonstration of the application is available in https://youtu.be/GzL0hwEu2OM.

Acrónimos

VRU	Vulnerable Road User
RSU	Roadside Unit
IPFS	InterPlanetary File System
VAM	Vulnerable Road User Awareness Message
MQTT	Message Queuing Telemetry Transport