Hybrid Evaluation Framework for Vehicle Edge Computing

Diogo José Guedes Marta

Licenciatura em Engenharia Electrotécnica e de Computadores
Departamento de Engenharia Electrotécnica

Instituto Superior de Engenharia do Porto

2019/2020

Este relatório satisfaz, parcialmente, os requisitos que constam da Ficha de Unidade Curricular de Projeto/Estágio, do 3º ano, da Licenciatura em Engenharia Eletrotécnica e de Computadores

Candidato: Diogo José Guedes Marta, Nº 1170907,

1170907@isep.ipp.pt

Orientação científica: Dr. Ricardo Severino, rarss@isep.ipp.pt,

CISTER Research Center

Co-Orientação: Dr.Pedro Santos, pss@isep.ipp.pt

Co-Orientação: Ênio Filho, enpvf@isep.ipp.pt

Licenciatura em Engenharia Electrotécnica e de Computadores

Departamento de Engenharia Electrotécnica

Instituto Superior de Engenharia do Porto

Agradecimentos

Gostava de agradecer aos meus orientadores no CISTER que me apoiaram e orientaram durante este projeto ao longo deste semestre e forneceram suporte para a elaboração deste relatório.

Quero agradecer ao meu colega e amigo Diogo Lopes que me acompanhou durante este período e pelo espírito de companheirismo que demonstrou.

Também quero agradecer aos meus amigos Catarina e Renato que viveram comigo grande parte da minha vida universitária.

Por fim, quero agradecer à minha família que deu o apoio emocional que muitas vezes precisava.

Abstract

With the rapid developments in communications towards 5G, topics such as the Internet of Things (IoT) and full automation of vehicles, Mobile Edge Computing (MEC) have become of high-interest in the scientific community. Nowadays many cloud services have become an important tool for modern society, but with the growth of connected mobile devices, cloud computing as we know will become incapable of providing the necessary service quality and it is here where MEC gains a particular interest.

In order to compare Mobile Edge Computing and Mobile Cloud Computing (MCC), this project aims at setting up a framework for joint simulation-real world for evaluation of communications between vehicles, edge and cloud nodes. To that end, we use Gazebo and ROS to simulate vehicle mobility and OMNeT++ to implement communications between simulated nodes and real nodes.

In a initial approach, we developed a system involving video streaming and an object recognition carried out to an edge/cloud node to compare round-trip time using MEC and MCC resources.

Next, we setup an OMNeT++ hybrid network connecting simulation world to real world to compare and evaluate MEC and MCC. The network is implement a Vehicle-to-Infrastructure (V2I) scenario.

Keywords: Mobile Edge Computing, Mobile Cloud Computing, OMNeT++, V2I, autonomous driving

Resumo

Com o rápido desenvolvimento da comunicações do 5G, temas como Internet of Things e carros cada vez autónomos, MEC tornou-se um tópico com alto interesse por parte da comunidade científica. Atualmente já há serviços cloud implementados na nossa sociedade e que rapidamente tornaram-se um elemento vital na nossa vida, mas com o crescimento de dispositivos conectados à rede, a computação cloud irá ter falhas e tornar-se-á obsoleta, e é a estes problemas que a MEC procura fornecer respostas.

Para comparar Mobile Edge Computing e Mobile Cloud Computing (MCC), este projecto visa a criação de uma framework de ambiente simulação-real para avaliar comunicações entre veículos e os nós edge ou cloud. Para esse fim , foi usado Gazebo e ROS para simular a mobilidade dos veículos e o OMNeT++ para estabelecer comunicações entre nós simulados e reais.

Numa primeira abordagem foi desenvolvido uma sistema que realiza transmissão de vídeo e reconhecimento de objetos para comparar a latência entre MEC e MCC.

Depois foi desenvolvido uma rede OMNeT++ híbrida que conecta o mundo simulado e real para comparar e avaliar MEC e MCC. Nesta rede foi implementado um cenário Vehicle-to-Infrastructure.

Palavras-Chaves: Mobile Edge Computing, Mobile Cloud Computing, OMNeT++, V2I, condução autónoma

Contents

List of Figures

Acronyms

CAM	–	Cooperative Awareness Message
ETSI	–	European Telecommunications Standard Institute
IoT	–	Internet of things
MCC	–	Mobile Cloud Computing
MEC	–	Mobile Edge Computing
RAN	–	Radio Access Network
RSU	–	Road Side Unit
ROS	–	Robot Operating System
TCP	–	Transmission Control Protocol
UDP	–	User Datagram Protocol
ICMP	–	Internet Control Message Protocol
V2V	–	Vehicle-to-Vehicle
V2I	–	Vehicle-to-Infrastructure
VDP	–	Vehicle Data Provider
RM	–	Robot Middleware
NED	–	Network Description
OMNeT++	–	Objective Modular Network Testbed in C++
VEC-HEF	–	Vehicular Edge Computing Hybrid Evaluation Framework

Chapter 1

Introduction

1.1 Overview

Nowadays, data collected by self-driving cars (video, localization, sensing) is increasing at a high rate. Despite the growing computational capabilities deployed in vehicles, there may not be capable to process all the information. Mobile Edge Computing (MEC) and Mobile Cloud Computing (MCC) are paradigms that are trying to address this problem for autonomous vehicles. They are based on distributing and processing data collected by multiple nodes on the network/cloud.[8]

Mobile Edge Computing is a new technology that offers a different and new options to mobile and vehicular applications. MEC provides similar cloud-computing solutions at the edge of network, so much closer to the end-user node. With this new model/paradigm, latency time will be reduced and will increase network efficiency when compared to Mobile Cloud Computing[9].

fairly Also with 5G, communications between the user node and the computing node will become faster and more reliable and this are relevant and decisive subjects when it comes to technologies such as MEC. The latency of Mobile Edge and Cloud Computing is based in two components: the transmission time between the two nodes and the processing time in computing node.

In this work, in the edge scenario, the infrastructure is based on vehicular node and a road-side unit (RSU) which is the computing node. In the other

hand, in the cloud scenario, the computing node is a standard cloud server provided by a Microsoft Azure service.

To better understanding and explore MEC, this thesis, presents a simulated environment integrating real-world connections, using several tools like e.g. OMNeT++, ROS or Gazebo, to evaluate and compare performance between Edge and Cloud Computing in vehicular applications.

This project was developed in CISTER – Research Centre in Real-Time and Embedded Computing Systems.

1.2 Research Objectives

The main objective of this project is to develop a hybrid(simulation and real-world components) environment based on Gazebo/ROS, OMNeT++ and edge/cloud nodes to test, evaluate and compare Mobile Edge and Cloud Computing technologies when applied to autonomous driving. To complement this is implementing an autonomous driving use case using object detection, using MEC resources, in order to integrate on a real environment.

This project's main research contributions are the following:

- Implementation of autonomous driving scenario in Gazebo to simulate autonomous car control with object detection
- Comparison of latency times between Edge and Cloud Computing with a video streaming where the computing node performs object detection
- Integrating of edge and cloud communications in OMNeT++ to evaluate perfomance in real-world situations

1.3 Document's Structure

This document covers six chapters. The following chapter will introduce Mobile Edge Computing's state of art and Vehicle Edge Computing technologies. Chapter 3 presents different technologies and platforms that have been used in this project. It is then followed by chapter 4, which demonstrates a first approach to Edge/Cloud Computing using video streaming with object detection, to compare latency between MEC and MCC. After

that chapter 5 presents a simulated platform using OMNeT++ to provide a more realistic data about the same matter to extend the results. Finally, Chapter 6 sums up the main conclusions and presents future work that can be realized in a near future.

Chapter 2

Overview of MCC/MEC and VCC/VEC

This chapter provides an overview over MEC/MCC and VEC/VCC.

2.1 Mobile Edge and Cloud Computing

MEC's concept was presented by the *European Telecommunications Standard Institute* (ETSI) [10] in middle of the decade. It is defined as "IT and cloud-computing capabilities within the Radio Access Network(RAN) in close proximity to mobile subscribers".

Until now, Mobile Cloud Computing (MCC) was the main paradigm when we talk about mobile computing but with the advances on Internet of Things (IoT) and 5G communications, MCC will become incapable of treat all this information due to massive increment of mobile device connected no network, so MEC will be essential to maintain a good and functional network. Some studies and companies, for example CISCO predicts, in 2023, there will be over 29 billions device connected to network [11].

The main principle in MEC is to receive data from a IoT device or, in the particular interest of this thesis, from a autonomous car and perform a processing task closer to the source when compared to MCC. MCC servers are highly centralized while MEC provides a more distributed service, in other words, a MCC server covers a large area with a bigger number of devices connect to him whereas a MEC server covers a reduced area with fewer

devices. MEC's applications focus on particularly in areas like augmented reality, video-analysis, autonomous driving and other sensitive tasks.

Because of proximity between the MEC servers and end users when compared to MCC, MEC will presented several advantages to the user and to overall system. In the following figure we can see the several differences between MEC and MCC.

Requirements/Features	Cloud Computing	Edge Computing
Latency	High	Low
Network Access Type	Mostly WAN	LAN(WLAN)
Server Location	Anywhere within the network	At the edge
Mobility Support	Low	High
Distribution	Centralized	Distributed
Task/Application Needs	Higher computation power	Lower latency
User Device	Computers, mobile devices (limited)	Mobile-smart-wearable devices
Management	Service Provider	Local Business
Number of Servers	High	Low
State	Soft and hard state	Soft state

Figure 2.1: Mobile Cloud Computing versus Mobile Edge Computing[1]

MEC has the main advantages of achieve lower latency, improving security for mobile applications and decrease bandwidth costs when compared to MCC. These advantages are briefly described in the following.

Lower Latency: Latency time is based in three components: propagation, communication and computation. In MEC, edge servers are commonly deploy near to the data source and normally the distance between user node and server are no longer than a few kilometers. In other side, Cloud Computing data centers can be hundreds of kilometers away. In terms of communication, MCC requires that information needs to pass several networks and with traffic congestion can provoke excessive delays of service, and that does not happen, or at least a much smaller scale, in Edge Computing. Finally in computation component, although Cloud servers have an enormous advantage due to a higher computational power than a Edge server,

a Cloud unit serves a much larger of devices, which can lead the lost of this advantage[8].

Security: MCC is based in a large data centers that serves a enormous quantity of devices at the same time, and so are more likely to cyber attacks. In other hand, MEC because a more distributed resources through the network are less susceptible to attacks.

Bandwidth: In MCC, with the convergence of information by several users node to cloud, the backbone transmission network becomes more and more congested and affects quality of transmission and increase bandwidth costs. At MEC because of a smaller number of users and smaller end-to-end distance, the congestion of network happens on a reduced probability.

Although MEC resources are growing in number and taking part in certain services that were previously taken by MCC, this does not mean that MEC will replace MCC, quite the opposite. These two paradigms correlate with each other and in many of the systems implemented today they use resources from both to get the best quality of service.

2.2 Vehicular Edge Computing

Vehicular Edge Computing (VEC) is the integration of MEC capabilities in vehicular networks in order to provide a nearest computing and communication resources to the vehicular users. The most interesting feature in VEC is its support fast mobility in vehicular networks (Figure 2.2),which demand a more dynamic topology and complex communications that do not occur in others MEC applications [12].

Autonomous cars are a highly complex systems and have many technologies associated to provide a secure driving including perception, sensing, decision making and localization resources. Although these vehicles have an exceptional computational power, it is not enough to process a huge amount data that comes from technologies mentioned before in real-time and take decisions based in information collected at the same time[13].

In order to guarantee the safety and also to improve efficiency, autonomous vehicles exchange intensive and latency-sensitive workloads for example video and other safety data with other vehicles in the same network towards edge servers, so that the vehicle system doesn't get overloaded with data and reduce the cost of self-driving cars. In VEC, this request for external computation resources can obtain the content that is needed from

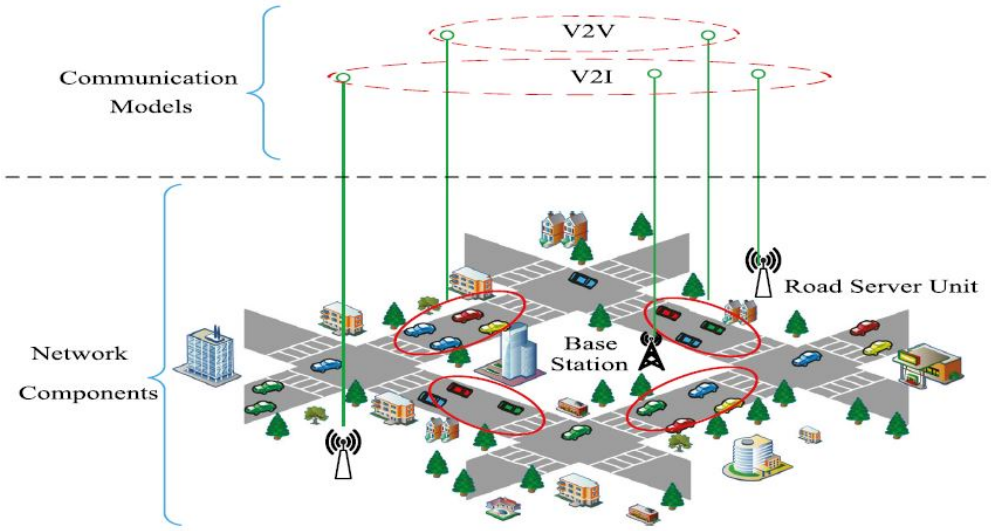

Figure 2.2: Architecture of Vehicular Networks[2]

near nodes instead of accessing cloud nodes, thus reducing end-to-end latency and improving efficiency of network bandwidth usage.

2.2.1 VEC's Architecture

The architecture of VEC [12] represented in Figure 2.3 is based in three layers, user layer where is vehicles nodes, MEC layer corresponded to RSU and cloud layer where is cloud computation resources.

2.2.1.1 Vehicular terminals

Vehicular terminals are mainly vehicles that incorporates several features:

- Sensing: vehicles collects information from the surrounding environment, i.e, cameras, sonars or geolocation systems.
- Communication: vehicles share information and communicate with others vehicles or RSU's using V2V and V2I communications.
- Computing:: in addition to transferring some computation tasks to the edge servers or the cloud servers for processing, vehicles can execute parts of the tasks that demands non-intensive computation.

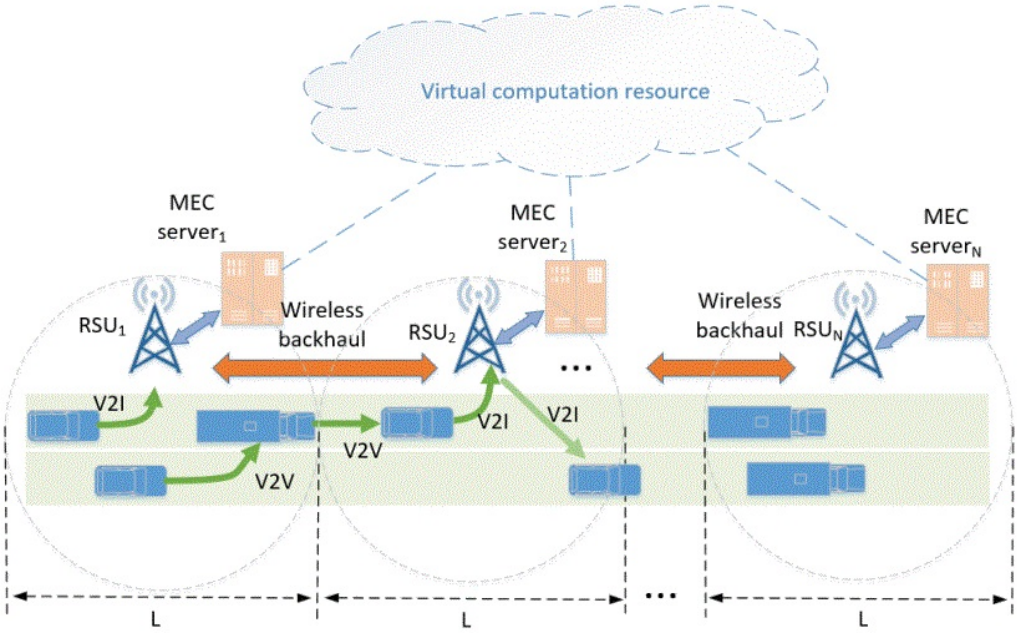

Figure 2.3: Architecture of Vehicle Edge Computing[2]

- Storage:the storage of vehicles can be used to cache popular contents for data sharing to improving system efficiency.

2.2.1.2 Road Side Units

RSU's are distributed around the city or region to cover maximum area possible and are responsible for receiving the information and sent to edge or cloud servers.

2.2.1.3 Edge and Cloud servers

Edge and Cloud servers have a very rich capacities in terms of computation and storage. Edge servers are designed for sensitive tasks where round trip time is a critical factor and cloud servers are more for heavy computation tasks.

Edge servers provide a distributed network while cloud servers set up a more centralized service and cover a much wider area when compared to edge servers. Cloud servers by getting connected to all edge servers that are within coverage area, they can get the uploaded information from them and based on that information provide a centralized control and a level management of network.

2.3 Integration of Simulated and Real Networks

OMNeT++ and the INET framework are commonly used to simulate IP-based networks and had several protocol implementations like TCP, UDP or ICMP. Over the recent years, INET framework have been used to create hybrid scenarios where simulated nodes communicate with real external nodes as the one that was developed in [3].

That project consist ofa Stream Control Transmission Protocol (SCTP) implementations with real external servers to evaluate, testing and further real implementations of SCTP model. SCTP is a protocol that combines UDP message-oriented feature and TCP congestion control with new concepts like streaming. To be able to establish communication between simu-

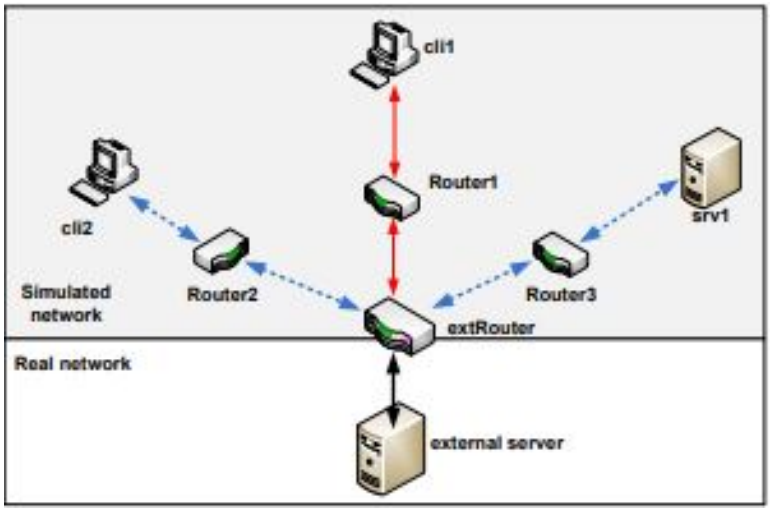

Figure 2.4: Network Framework[3]

lated network and a real IP-based server, it was created a route throughout a external interface. The most interesting feature about this implementation is that *extRouter* on Figure 2.4 belongs both to the real world and the simulated network.

Chapter 3

Technologies and Simulation Platforms

In this chapter presents different technologies and simulation platforms used in this project. It starts with an overview of ROS (Robot Operating System), after this there is a overview over COPADRIVe, a simulation environment in which this project is based.

3.1 Robot Operating System(ROS)

ROS [14] is an open-source framework , created in 2007 and since then had a fast growth and its been used for various applications. Due to to an extensive variety of services, like for example hardware abstraction or low-level device control, and also provides tools, libraries and conventions to auxiliary creation of robotic applications and to promote the reusing of code and sharing throughout the ROS community. It also very useful to running applications in several machines.

ROS is based in publisher-subscriber model, as shown in Figure 3.1, each ellipse represent a ROS node that can be subscriber or publisher. A publisher creates a topic with a defined message type, each node in network can subscribe the same topic and get the information that contains. This type of system improves network scalability and enables a more dynamic network by the publisher does not send information directly to one node but to network in general and allow that each node in same network access the topics and messages as needed. ROS has three level of concepts: the

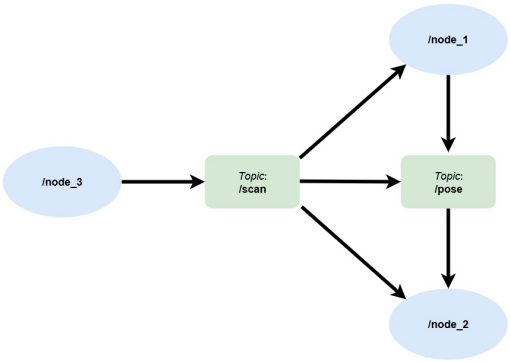

Figure 3.1: Publisher and Subscriber Example[4]

Filesystem level, Computation Graph level and the Community level, all are summarized in [15].

The Filesystem Level covers all resources of ROS that can be found on disk and the most common used are Packages, Messages and Services. Packages are the fundamental unit of ROS, they can contain ROS runtime processes (nodes), a ROS-dependent library, datasets, configuration files, or anything else that is usefully organized together. The Messages description define and store the messages used in each package. Service descriptions define the request and response data structures for services in ROS.

The Computation Graph level is the key to established the connection between ROS processe that are computing data together. Following are explaining each basic concept of this level:

- Nodes - Nodes are processes that perform computation and each node can communicate with other node by using distinct ROS concepts including topics.Each Robot normally have many nodes, and each node can be responsible for a certain functionality of robotic system.[16]

- Master - The main function of ROS Master is provide naming and registration services to the rest of the nodes in the ROS system. ROS Master is responsible to enable the communication peer-to-peer between a publisher and a subscriber who's interested in the topic that is been published.[17]

- Topics - Topics are intended for unidirectional, streaming communication in which nodes exchange messages. Each topic can be subscribed or published by simultaneous nodes, as well one node can publish or

subscribe to several topics. Nodes, generally, are not interested in with whom they are communicate, instead nodes that generate data publish into the topic and nodes that are interested ind data subscribe the topic.[18]

- Messages - Messages are a data structure that supports standard types like integer, string or boolean. Nodes use messages for publishing data to topics in order to communicate with another topic.[19]
- Bags - Bags are a format for saving and playing back ROS message data. When is difficult to collect some data that is necessary for develop algorithms, Bags are a very crucial mechanism to storage data.[20]

The last level of ROS is the Community level, who is responsible for allowing the share of resources and knowledge between users. This level provides different distributions, repositories and a wiki. In this project has been used ROS Melodic distribution[15].

3.2 Gazebo

Gazebo is a 3D robot-simulator created by Open Dynamics Engine(ODE) that makes it possible to test algorithms, design robots and environments, perform regression testing and also train AI system using realistic scenarios This tool needs to be integrated in a ROS system in order to control the simulated environment throughout a specific packages that made connection between ROS and Gazebo environment.

In this project, was used an simulated environment already created in COPADRIVe project, that will be presented next featuring three vehicles that communicate with each other that performs a full-functioned platooning driving system.

3.3 OMNET++

OMNET++(Objective Modular Network Testbed in C++) is an extensible, modular, component-based C++ simulation library and framework, designed for building network simulators. Network englobes wired and wireless communication networks, on-chip networks and queueing networks. Model frameworks of OMNET++ provides domain-specific functionality essentialy

support sensor networks, wireless ad-hoc networks, Internet protocols, performance modeling and photonic networks.[21]

OMNeT++ provides a component architecture for models. These components are based in C++ and then assembled into larger components and models using NED language.

Over the years, OMNET++ has gain a considerable popularity between scientific community and made acessible new models for diverse areas. In this project the models that were used are INET, Veins, Vanetza and Artery.

INET framework [22] is a model library of OMNET++ that provides protocols (TCP, UDP, IPv4, etc.), models and wired and wireless interfaces like Ethernet and IEEE 802.11. It is used widely for vehicular networks, LTE scenarios and designing and evaluating new protocols, also is a base for others simulation frameworks i.e. SimuLTE.

Veins framework[23] is a framework for running vehicular network simulations. It is based in OMNeT++, responsible for network simulation, and SUMO, a road traffic simulator.

Vanetza [24] is an implementation of the ETSI C-ITS protocol suite that provides protocols and features such as Basic Transport Protocol (BTP) and support for ASN1 messages like CAM.

Artery[25] is a extension from Veins framework but can be used independently also can use some features from Vanetza framework. Artery enables V2X simulations based on ETSI ITS-G5 protocols like GeoNetworking.

3.4 COPADRIVe

As previously mentioned, this project used an already implemented simulation environment described in COPADRIVe project [5]. In COPADRIVe (Figure 3.2), the main objective was to implement a platooning driving system where cars communicate by sending several informations like position or speed of leader car, to each other to adjust the behavior of following cars according to information send by leader car.

This project is based in a ROS/OMNeT++ integration, where ROS and Gazebo are responsible for controlling and the movement of cars, with OMNET++ to implement a simulation system that provides communications between cars.

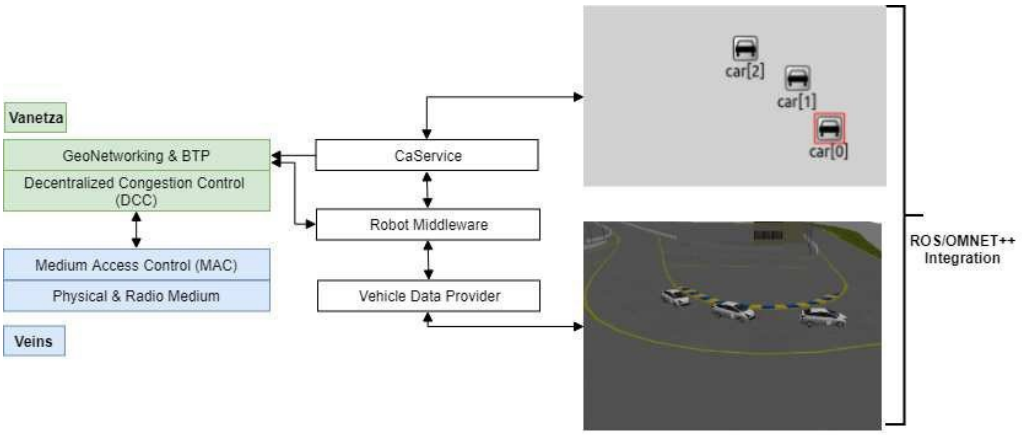

Figure 3.2: COPADRIVe Framework Architecture[5]

3.5 Microsoft Azure

Microsoft Azure is a cloud computing service create in 2010 and provides several tools including virtual machines (VM), SQL databases and App Services.

Azure is based in different cloud servers across the world and have different types of VM in order to satisfied all the needs of community. In this project has been used a Standard B2s VM, that has a 2 vcpus and 4 GiB memory, located at Texas in South Central region of United States of America.

Chapter 4

Testbed for Cabled Component of VEC Applications

This chapter presents a first approach to evaluate MEC/VEC perfomance mainly in terms of round-trip time. We developed a simulation environment using video streaming to a external unit, where has been executed a object detection using YOLOv3.

The main benefit of MEC is a reduced latency on end-to-end transmission when compared to a cloud service. Therefore to really understand the difference of MEC and MCC, in this project we setup and tested a video streaming and object detection application with both paradigms. Real-time object detection in self-driving cars requires a high computational power that some systems cannot provide, consequently, the need for transmitting these processing tasks to a external node in network appears. Nowadays, this node corresponds mainly to a cloud server far away from the vehicle but with the predicted bottleneck of this servers in a near future, this cloud applications are migrating and decentralise to a near location from user to reduce round trip time of transmission and improving the quality of service.

4.1 Target Application

Object recognition is a process related to computer vision that detects and differentiates objects in digital images. In autonomous driving, the interaction between human and car behavior pretends to be the minimum as

possible and for that car must perceive the conditions or road signs around it. This process demands a high computation ability of vehicle and for this reason, increased latency time during image processing can provoke unsafe decisions by onboard computer. In order to resolve, or at very least to minimize the probability of these errors occurs, object recognition is often connected to machine learning tasks.

YOLOv3 is a Real-Time Object Detection system developed by Joseph Redmon at University of Washington[6]. In Figure 4.1, it is shown an example of such application in a real situation.

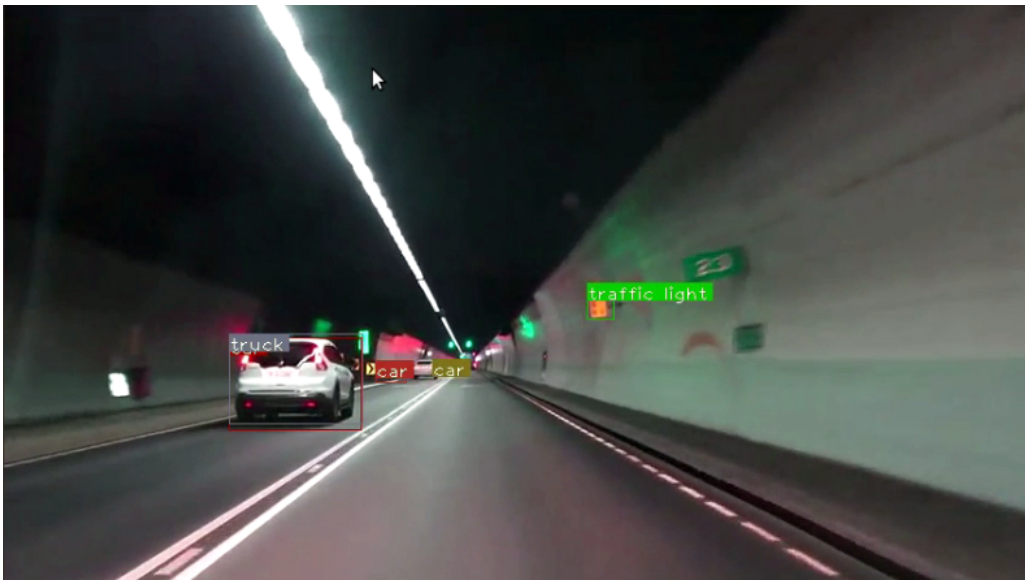

Figure 4.1: Object recognition in a real image

This system is based in 53 layer fully convolutional underlying architecture and its divided in stages in order to a correct object identification. In first instance, YOLOv3 predicts bounding boxes using dimension clusters as anchor boxes and then predicts an objectness score for each bounding box. After this each box predicts the classes that the bounding box may contain using multi label classification. Theses boxes are predicted in three different scales and together with a feature extractor and several convolutional layers creating DarkNet-53 neural network. DarkNet-53 performs better and delivered a faster response when compared to others networks that are common used in object recognition[6]. As 4.2 show, YOLOv3 had a less inference time when compared to other models.

In this project, YOLOv3 has been used to detect particularly road signs

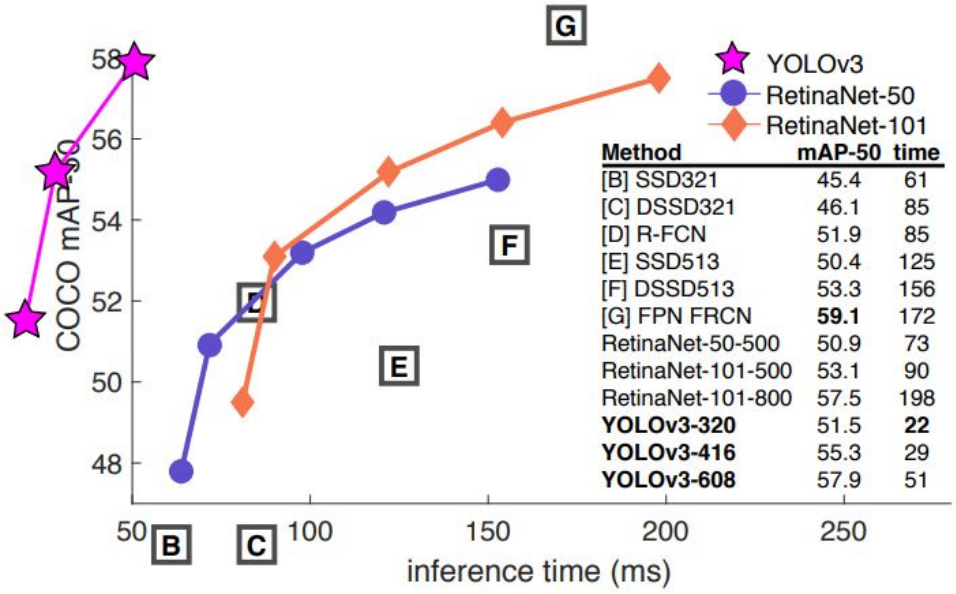

Figure 4.2: Object recognition models[6]

but also detects other variants like car or people. This processing task controls vehicle in order to stop the car when find a STOP sign. In next figure, it is possible to notice to see object detection in a simulation environment.

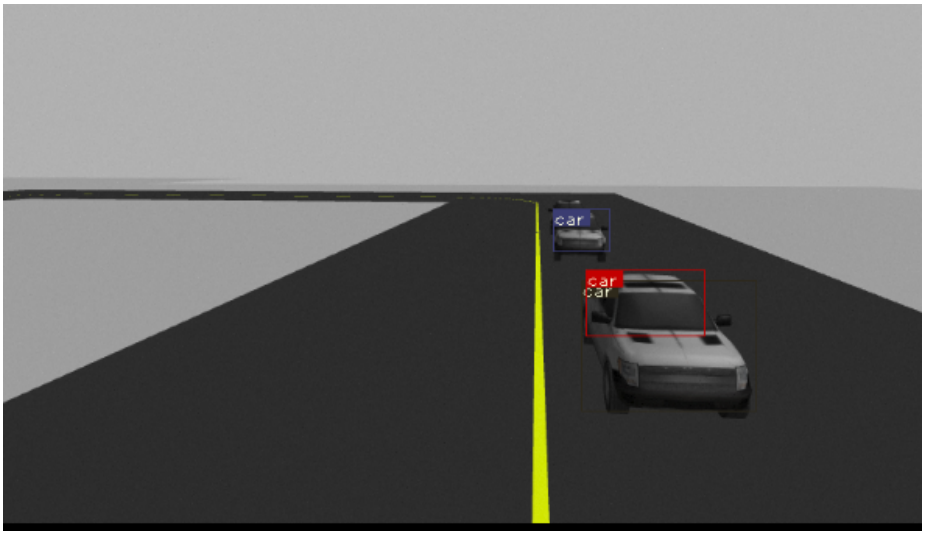

Figure 4.3: Object recognition in a simulated environment

4.2 Architecture of Developed Testbed

As a first approach to this project, it was implemented an introductory experience to have a first point for evaluating MEC/VEC.

As we can see in Figure 4.4, it is developed a simulation environment on local machine with Gazebo. This Gazebo simulation is provided by COPADRIVe project where is created a track with an autonomous car and several other cars as obstacles and road signals.

Gazebo uses a meta-package called *"gazebo_ros_pkgs"* [26], these packages provide wrappers around Gazebo stand-alone simulation to create necessary interfaces to simulate a robot model using ROS messages and services. Gazebo is responsible to publish topics with data from the car and with particular interest a ROS topic called *"/car1/front_camera/image_raw"*, with a frequency of 30Hz, that contains a video type message from the front view of vehicle that will be transmitted to a external sever.

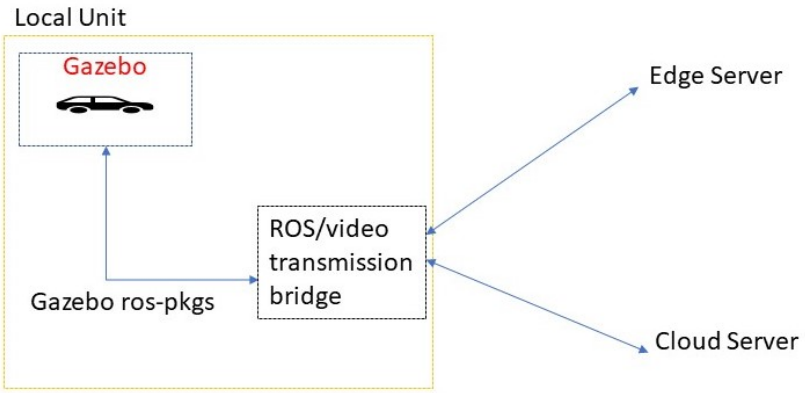

Figure 4.4: Software architecture of project

Communications between local machine and server node can be done in two ways in this framework: running ROS over multiple machines in the same network, sharing topics, messages and services between them, similar to a master-slave system, or subscribing the topic with data necessary and send data over a socket.

Due to incompatibility of sending video messages based on ROS via socket to a external unit on a different network, i.e cloud server, then, in the same unit where Gazebo is implemented, it has been set up a ROS/video-transmission bridge, demonstrated in Figure 4.6. This bridge was build in

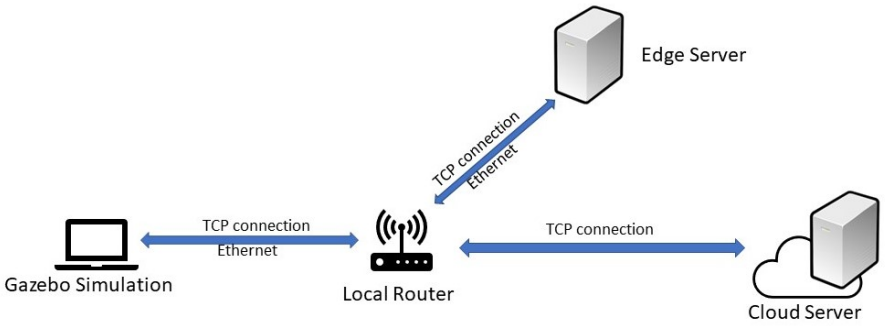

Figure 4.5: Physical architecture of project

order to convert ROS topics video-type message to a frame and to transmitting for other units. This bridge subscribed the topic */car1/front_camera/image_raw*.

This bridge is also required because YOLOv3 cannot process image directly from ROS, so even if there is master-slave system sharing a ROS message between machines in the same network this bridge (Figure 4.6 it will always be necessary. This bridge uses a ROS library called *"Cv_bridge"*, converting a ROS image into a OpenCV image by encoding. In this bridge was used *"bgr8"* encoding. We decided to use this type of encoding because we also tried different types of encoding but it was *"bgr8"* that presented more quality.

```
if __name__ == "__main__":

        def callback(data):
                frame = bridge.imgmsg_to_cv2(data,'bgr8')

                data = pickle.dumps(frame)
                message_size = struct.pack("L", len(data))
                clientsocket.sendall(message_size + data)
                now=datetime.now()
                time1=now.strftime("%S.%f")
                time1=float(time1)
                byte_recv= clientsocket.recv(1024)
                now=datetime.now()
                time2=now.strftime("%S.%f")
                time2=float(time2)
                label=byte_recv.decode()
                msg=String()
                msg.data=label
                pub.publish(msg)


        rospy.Subscriber("/car1/front_camera/image_raw",Image,callback)
        rospy.spin()
```

Figure 4.6: Code of bridge between ROS and socket

After that, local unit send to edge or cloud server via a Transmission

Control Protocol (TCP) connection. In the server node is performed a object recognition task, that will be explained with more detail in next section, and then informs local unit if detects any obstacle or not.

As mentioned before in this thesis, cloud server is based in a Microsoft Azure virtual machine located in Texas. This cloud service is a Standard B2s that incorporates 2 VCPUs and 4GB of RAM. The edge node is a Intel i3-4005U CPU and 4GB of RAM memory running Ubuntu 18.04 and client unit is also a laptop with a Intel i7-8565U CPU and 8GB of RAM running Ubuntu 18.04.

4.3 Comparison of VEC/VCC using the Developed Testbed

To measure round-trip delay time in each case was considered two components: **processing time**, the total time for data to be processed in server unit, and **transmission time**, the amount of time from the client send frame and receive return of the same frame less the amount of time for processing task. For each case, VEC or VCC, were performed five tests and in each one were sent about 750 frames. In Figure 4.7, show round-trip time of communication between the user node and processing node. In the same figure, red bar corresponds to transmission time while blue bar corresponds to processing time.

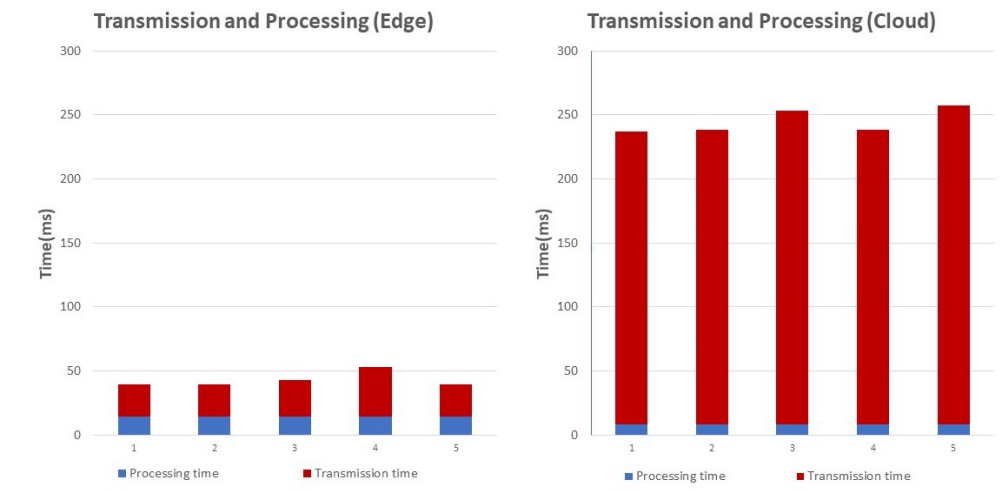

Figure 4.7: Latency time in Edge and Cloud Computing

In VEC round-trip time is clearly lower than VCC. On VEC, transmission time averaged a 28.65 milliseconds and processing time averaged 14.42 ms which gives a total round-trip time of 43.07 ms. On VCC, round-trip time averaged 245 ms which is 5.7 times bigger than Edge round-trip, but the mean processing time is lower taking 8.22 ms, demonstrating higher computational power of Cloud unit, in other hand, transmission time is about 236.77 ms which is by far greater than VEC's transmission time.

In figure 4.8, we can see the box plots of round-trip times and as in the previous figure we can notice that latency values in Edge Computing are considerable lower than when Cloud Computing implemented. Also can be noticed Cloud Computing application had a higher dispersion of data.

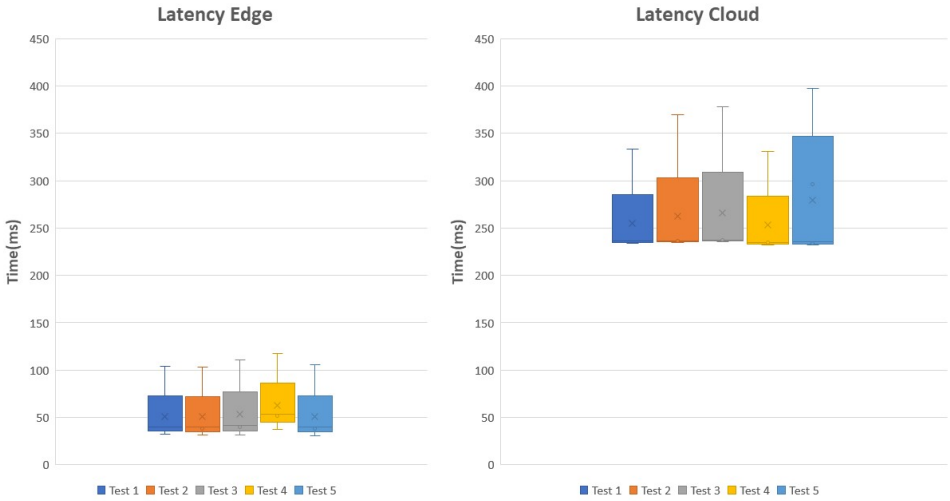

Figure 4.8: Distribution of round-trip times

4.4 Final Remarks

With the development of this testbed, it is possible to notice that VEC, in terms of round-trip time, demonstrates some advantage over VCC.

Although a cloud server provide a better and faster unit in terms of processing, transmission time is clearly a negative point when compared to a edge server.This was a first perspective of VEC and VCC differences, nevertheless this is not a perfect testbed that totally differentiates VEC and VCC.

Should be noted that the fact of edge server be in the same network connected by cabled with computer where Gazebo was performed, limited this testbed in terms of realism.

Chapter 5

VEC Hybrid Evaluation Framework

In this chapter is presented a simulation environment using OMNeT++ integrating ROS and Gazebo in order to compare round-trip time between Vehicular Edge Computing and Vehicular Cloud Computing.

The main goal of this project was to evaluate round-trip time in a typical scenario of VEC/VCC like in Figure 5.1, where car1 sends information to a near RSU and then to a edge/cloud server. The server process the data and sends back to car2 through RSU.

Given the difficulty in accessing an actual vehicle and RSU, further aggravated by the pandemic situation, we developed a Hybrid Evaluation Framework for Vehicular Edge Computing (VEC-HEF). This hybrid framework merges simulation world and real world to measure and evaluate round-trip time in VEC and VCC. It should be mentioned that, at this time, full integration between the real-world and simulation parts is still undergoing final stages of development.

5.1 VEC-HEF Architecture

As we can see in Figure 5.2 this project it is divided in 2 parts: a simulation environment where OMNeT++ is integrated with Gazebo/ROS and real world component. In the simulated component there is two cars and one

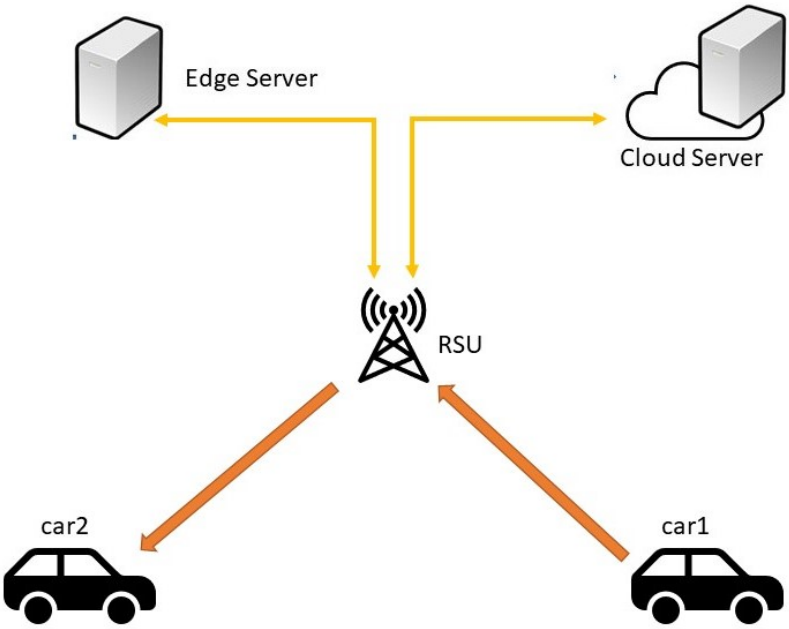

Figure 5.1: VEC/VCC scenario

RSU. The communication between cars and RSU is made by CAM using ITS-G5[27]. This RSU makes the connection between simulation world and real world. The real world part is based on the components that were part of the previous chapter namely edge server and cloud server with the same characteristics as before.

As mentioned before, this work already takes advantage of the framework created in COPADRIVe[5] mainly how vehicles communicate with each other. Until now, each car reads several characteristics of his movement that were obtained from ROS/Gazebo system and send the same information to other cars via wireless by using a Cooperative Awareness Message (CAM) in OMNeT++.

This project is formed by three components:

- Gazebo - responsible for the movement of cars and the data generation;
- OMNeT++ - establishing communications between cars and RSU;
- Server nodes - provide edge and cloud resources;

The main effort of development took place in OMNeT++, given that we wanted to connected the simulated vehicular scenario with the real-world

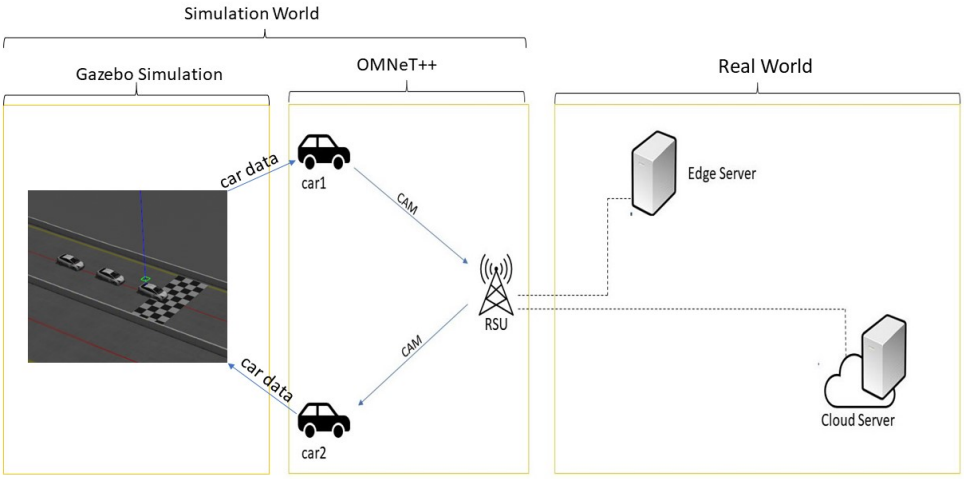

Figure 5.2: VEC Hybrid Evaluation Framework

components of the scenario that were described in the previous chapter. In COPADRIVe project was designed a platooning driving system, that for this work was changed to have a structure similar to the scenario identified in the Figure 5.1.

5.2 Structure of OMNeT++ Modules

To better understand the following discussion we first introduce some of the underlying concepts of OMNeT++ simulations. OMNeT++ uses Network Description(NED) language[28] to allows the user to declare modules which will make up the network. An OMNeT++ network (Figure 5.3) consists in modules which communicate between them with messages. These modules can be defined as simple modules or compound modules. Simple modules are a active component in model and they are defined by listing its owns parameters and gates. A simple module can be extended by others modules and shares with them is own parameters and gates. Compound modules are modules composed of one or more submodules that can be either simple or compound modules. The submodules and the connections within that module are specified by compound module description. Each module can connect with others by gates allowing then to send messages to modules in that network.

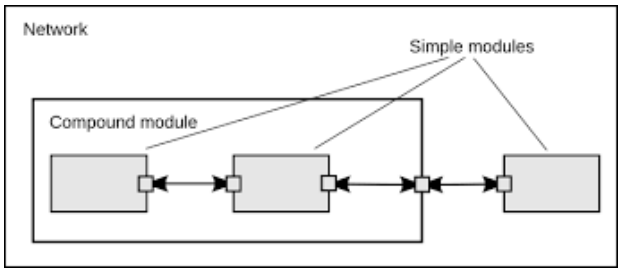

Figure 5.3: Model structure in OMNeT++[7]

5.3 Connection between to Simulated and Real-World Scenario Parts

There are several ways to connect the two worlds with a external interface and tested different ways to connect both "worlds" using OMNeT++ network and external interfaces. In [29] three alternatives scenarios to established communications between two worlds are presented: socket connection, source code integration and shared library integration. OMNeT++ himself had some samples from hybrid examples, namely *"ExtTelnet"* module represented in figure 5.4. This example is the starting point to implement on COPADRIVe framework a connection to real world. The network of this example has four clients, where one of them is a real-world unit, that is connected to IP of machine and exchanges messages between them throughout server and cloud modules.

Initially, to better understanding and to see if this is a viable option to follow, we changed how this framework works to have a similar functionality to what we intend to implement on COPADRIVe framework. We sent directly information provided by one client module to external client and analyzed if this solution could be migrated to the framework we desire. This approach was also very appealing to test since there is a certain analogy with the project that we wanted to develop.

One of main problems of connecting real world to simulation world with OMNeT++ is scheduling and synchronization real time with simulated time. OMNeT++ performs a time discrete simulation based on CPU properties[29], which does not match with real time provoking a desynchronization between real and simulation world. To overcome this problem was been used *"cSocketRTScheduler"* library, that was implemented in example mentioned above. This library is responsible not only to establish socket

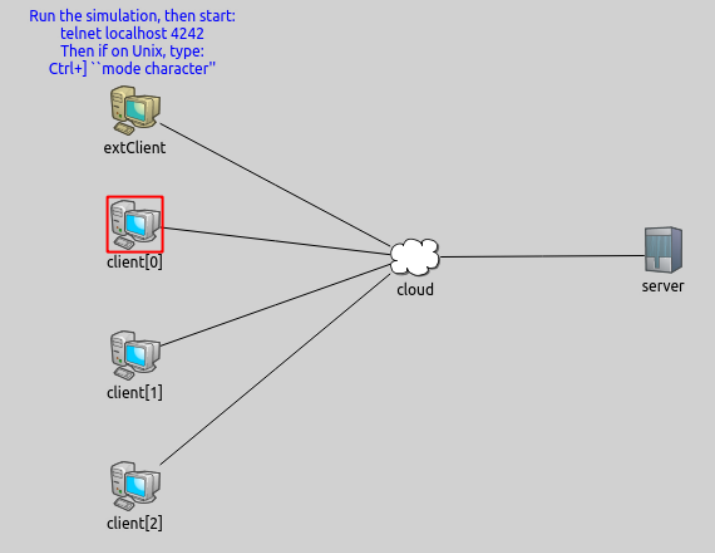

Figure 5.4: ExtTelnet Design

```
void cSocketRTScheduler::setupListener()
{
    listenerSocket = socket(AF_INET, SOCK_STREAM, 0);
    if (listenerSocket == INVALID_SOCKET)
        throw cRuntimeError("cSocketRTScheduler: cannot create socket");

    int enable = 1;
    if (setsockopt(listenerSocket, SOL_SOCKET, SO_REUSEADDR, (const char*)&enable, sizeof(int)) < 0)
        throw cRuntimeError("cSocketRTScheduler: cannot set socket option");

    sockaddr_in sinInterface;
    sinInterface.sin_family = AF_INET;
    sinInterface.sin_addr.s_addr = INADDR_ANY;
    sinInterface.sin_port = htons(port);
    if (bind(listenerSocket, (sockaddr *)&sinInterface, sizeof(sockaddr_in)) == SOCKET_ERROR)
        throw cRuntimeError("cSocketRTScheduler: socket bind() failed");

    listen(listenerSocket, SOMAXCONN);
}
```

Figure 5.5: cSocketRTScheduler socket establishment

connection during the initialization of network (Figure 5.5) and synchronize both worlds. To create the interface (Figure 5.6) is called a function *"setInterfaceModule"* from the class *"ExtInterface"*.

```
void ExtTelnetClient::initialize()
{
    cout << "extTelnetClient initialize" << endl;
    rtEvent = new cMessage("rtEvent");
    rtScheduler = check_and_cast<cSocketRTScheduler *>(getSimulation()->getScheduler());
    rtScheduler->setInterfaceModule(this, rtEvent, recvBuffer, 4000, &numRecvBytes);

    addr = par("addr");
    srvAddr = par("srvAddr");
}
```

Figure 5.6: *Extclient* initialize function

5.4 Implementation of Simulated Vehicular Scenario

This OMNeT++ network represented in Figure 5.8 is composed by three cars modules, where one of them is a stationary station and takes the role of a RSU, and one module that connects simulation environment with a external server. As mentioned before it is intended that car1 sends his own data to RSU, then RSU send the same information to a external server and waits for reply from server and after that RSU sends data to car2.

Each car is a compound module (Figure 5.7) composed by other modules like i.e Middleware, Vehicle Data Provider(VDP) and Robot Middleware(RM) that are responsible for sharing data between cars generated by Gazebo simulator. VDP is the bridge that supplies RM with data provenient from the Gazebo simulator by subscribing the topics that contains data of cars. RM then uses this same data to fill ITS-G5 CAM's data fields (i.e Speed values) through the Cooperative Awareness Service (CaService) that proceeds to encode this data fields in order to comply with ITS-G5 ASN-1 definitions. RM also provides GPS coordinates to define the position of the nodes in the INET mobility module[5].

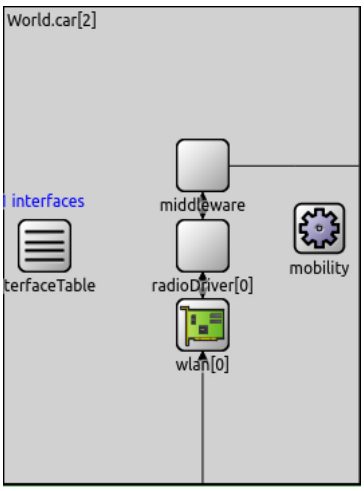

Figure 5.7: Car/RSU compound module

Initially the RSU was based in a predefined module but some difficulties appeared specially in terms of communication more precisely receive messages from the cars due to the structure of the message itself that needed certain data that were provided by Gazebo and that caused malfunctions

of the network. Because of that, instead of using a specified module for RSU, we set one of the cars to function like a RSU. In this case, that car instead of having a normal behavior and be part of platooning system, it is now a stationary unit providing a new way of communications in framework. Currently, there is no integration with external edge or cloud node. Hence, current operation is as follows. When car1 emits data to RSU, data is stored until receive a reply from server. Then when RSU sends to car2 the stored data. Normally, the RSU would send its own vehicle data in the CAM message (obtained through the VDP module), we altered this to obtain the behaviour described above.

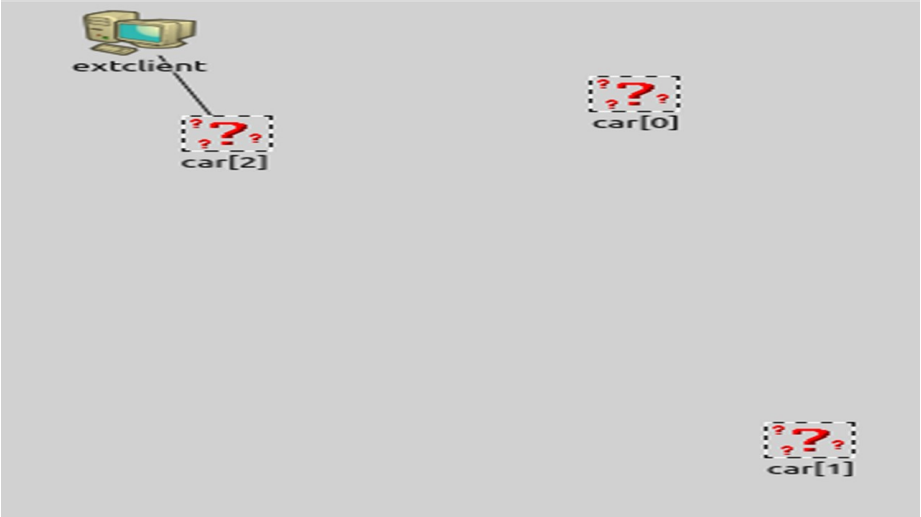

Figure 5.8: Hybrid System Design

Besides that RSU module is also connected to *Extclient* module by a wired connection. It is in *Extclient* module where external interface is configured using *"cSocketRTScheduler"* library. However during the developing of this project was not possible to establishing a link between external server, the socket was created and can be reached by other machine however we are currently observing an error when *Extclient* module sends a packet through the established socket.

5.5 Final Remarks

Although we have not been able to test the entire end-to-end communication (simulated vehicle1 to simulated RSU, to real-world edge or cloud node, and back to simulated RSU, and ultimately to simulate vehicle 2),

it was possible to simulated the trajectory of data throughout the framework independently, i.e., in the real-world (Chapter 4) and in the simulated vehicular scenario (described in this Chapter).

Chapter 6

Conclusion and Future Work

A overall conclusion of project and results obtained during the semester. Presentation of future work that may be carried out in continuation of this project.

6.1 Conclusion

Initially was idealized a solution to demonstrate MEC capabilities on a real robotic due to the restrictions imposed by the pandemic, the approach changed to simulation. Despite that, the main objective remained equal which was to demonstrate differences in round-trip time between MEC and MCC.

In the first approach was possible to compare round-trip time in MEC and MCC an it is noticed that MEC had a considerable lower round-trip time leading to a better overall perfomance of system. It should be also mentioned that YOLOv3 does not shows the same level of precision on the Gazebo simulator when compared to a real video.

In the hybrid system with OMNeT++ network was not possible to collect data that was pretended although the framework is builded and functional at least in communication between cars and RSU. Meanwhile will be tried to solve the problem mentioned in chapter 5 in time for presentation.

6.2 Future Work

As future work, could be to go back to the initial idea and develop a system as it was presented in the project in testbed and maybe implement a new functionality in the control of an autonomous car using MEC properties, or continue the project in a simulation platform and improving the control of platooning system.

Bibliography

[1] A. C. Baktir, A. Ozgovde, and C. Ersoy, "How can edge computing benefit from software-defined networking: A survey, use cases, and future directions," *IEEE Communications Surveys Tutorials*, vol. 19, no. 4, pp. 2359–2391, 2017. [cited on p. xiii, 6]

[2] L. Li, H. Zhou, S. X. Xiong, J. Yang, and Y. Mao, "Compound model of task arrivals and load-aware offloading for vehicular mobile edge computing networks," *IEEE Access*, vol. 7, pp. 26 631–26 640, 2019. [cited on p. xiii, 8, 9]

[3] I. Rüngeler, M. Tüxen, and E. P. Rathgeb, "Integration of sctp in the omnet++ simulation environment." [cited on p. xiii, 10]

[4] Get started with ros 2. [Online]. Available: https://www.mathworks.com/help/ros/ug/get-started-with-ros-2.html [cited on p. xiii, 12]

[5] B. Vieira, R. Severino, E. V. Filho, A. Koubaa, and E. Tovar, "Copadrive - a realistic simulation framework for cooperative autonomous driving applications," Nov 2019. [cited on p. xiii, 14, 15, 26, 30]

[6] J. Redmon and A. Farhadi, "Yolov3: An incremental improvement." [cited on p. xiii, 18, 19]

[7] A. Varga and G. Pongor, "Flexible topology description language for simulation programs," 09 2020. [cited on p. xiii, 28]

[8] Y. Mao, C. You, J. Zhang, K. Huang, and K. B. Letaief, "A survey on mobile edge computing: The communication perspective," *IEEE Communications Surveys Tutorials*, vol. 19, no. 4, pp. 2322–2358, 2017. [cited on p. 1, 7]

[9] M. Patel, D. Sabella, N. Sprecher, and V. Young, "Contributor, Huawei, Vice Chair ETSI MEC ISG, Chair MEC IEG Working Group," p. 16. [cited on p. 1]

[10] "Mobile-edge computing—introductory technical white paper." [Online]. Available: https://portal.etsi.org/Portals/0/TBpages/MEC/Docs/Mobile-edge_Computing_-_Introductory_Technical_White_Paper_V12018-09-14.pdf [cited on p. 5]

[11] "Cisco annual internet report (2018–2023) white paper." [Online]. Available: https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.html [cited on p. 5]

[12] L. Liu, C. Chen, Q. Pei, S. Maharjan, and Y. Zhang, "Vehicular edge computing and networking: A survey." [cited on p. 7, 8]

[13] S. Liu, L. Liu, J. Tang, B. Yu, Y. Wang, and W. Shi, "Edge computing for autonomous driving: Opportunities and challenges," *Proceedings of the IEEE*, vol. 107, no. 8, pp. 1697–1716, 2019. [cited on p. 7]

[14] Ros introduction. [Online]. Available: http://wiki.ros.org/ROS/Introduction [cited on p. 11]

[15] Ros concepts. [Online]. Available: http://wiki.ros.org/ROS/Concepts [cited on p. 12, 13]

[16] Ros nodes. [Online]. Available: http://wiki.ros.org/Nodes [cited on p. 12]

[17] Ros master. [Online]. Available: http://wiki.ros.org/Master [cited on p. 12]

[18] Ros topics. [Online]. Available: http://wiki.ros.org/Topics [cited on p. 13]

[19] Ros message. [Online]. Available: http://wiki.ros.org/Messages [cited on p. 13]

[20] Ros bags. [Online]. Available: http://wiki.ros.org/Bags [cited on p. 13]

[21] Omnet++. [Online]. Available: https://omnetpp.org/intro/ [cited on p. 14]

[22] What is inet framework? [Online]. Available: https://inet.omnetpp.org/Introduction.html [cited on p. 14]

[23] C. Sommer, R. German, and F. Dressler, "Bidirectionally Coupled Network and Road Traffic Simulation for Improved IVC Analysis," *IEEE Transactions on Mobile Computing (TMC)*, vol. 10, no. 1, pp. 3–15, January 2011. [cited on p. 14]

[24] Vanetza in a nutshell. [Online]. Available: https://www.vanetza.org/ [cited on p. 14]

[25] Artery - v2x simulation framework for etsi its-g5. [Online]. Available: https://omnetpp.org/download-items/Artery.html [cited on p. 14]

[26] Tutorial: Ros integration overview. [Online]. Available: http://gazebosim.org/tutorials?tut=ros_overview [cited on p. 20]

[27] ETSI, "Etsi tr 102 638." [cited on p. 26]

[28] OMNeT++, "Simulation manual." [cited on p. 27]

[29] C. P. Mayer and T. Gamer, "Integrating real world applications into omnet++." [cited on p. 28]