FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Simulation-based Evaluation of Edge Computing-Assisted Applications

André Cipriano Sousa

MASTER THESIS

U.PORTO

FEUP FACULDADE DE ENGENHARIA UNIVERSIDADE DO PORTO

Master in Electrical and Computers Engineering

Thesis Advisor: Pedro Miguel Santos

July 31, 2020

Simulation-based Evaluation of Edge Computing-Assisted Applications

André Cipriano Sousa

Master in Electrical and Computers Engineering

July 31, 2020

Resumo

Nos últimos anos, a Computação Limítrofe (*Edge*) para Aplicações Veiculares emergiu como uma solução para solucionar o problema de providenciar serviços para a componente de computação infrastrutural das aplicações veiculares de uma forma rápida e eficaz e para acelerar o aparecimento de Veículos Inteligentes. Este paradigma consiste em utilizar as crescentes capacidades computacionais e de armazenamento disponíveis pelos dispositivos no limiar das infrastuturas da Rede, as quais estão próximas dos utilizadores finais, como complemento aos serviços da Computação em Nuvem. O crescimento exponencial da quantidade de informação a processar, o aumento da sensibilidade temporal das aplicações e a elevada mobilidade dos veículos, constituem grandes desafios em situações reais que necessitam de ser abordados para a Computação Limítrofe atuar de forma adequada. Desta forma, surge a necessidade de se desenvolverem métodos mais eficazes para gerir a infrastutura da Rede e da Edge para providenciar serviço contínuo aos clientes veiculares.

Na tese proposta, é apresentada uma nova heurística baseada em "Classificação", que tem em consideração estados anteriores do uso da rede de forma a encontrar formas de alocar as tarefas veiculares progressivamente mais eficientes, em cada momento de decisão. Quando comparado com a heurística de Recozimento Simulado, foi conseguido a obtenção valores de latência menores. Para uma aplicação de processamento de imagem, os valores da *performance* de latência observados usando a heurística de "Classificação", quando comparado com os de Recozimento Simulado, foi humilde: até 2.5%. Ao comparar o número de tarefas veiculares servidas com sucesso, a heurística proposta também foi ligeiramente melhor, em 3.5%. Numa diferente aplicação veicular, focada mais em segurança e com menores requesitos de processamento, foram observados valores preliminares de *performance* de latência até 5%, bem como um aumento até 10.1% no número de tarefas servidas com sucesso.

Ambas as heurísticas foram implementadas numa nova Estrutura de Simulação, desenvolvida com o propósito de poder acomodar métodos diferentes de solucionar o problema desta tese, e desta forma, pode ser facilmente preparada para se adaptar a melhoramentos da heurística proposta, ou para o desenvolvimento de outras soluções. Pode inclusive ser usada como uma plataforma de estudo de outros problemas, e esperamos que possa ser uma contribuição para a comunidade que trabalha em comunicações veiculares e Computação Limítrofe. Houve um cuidado considerável para garantir que todos os valores usados fossem baseados em aplicações reais, *hardware* de mercado e fatores de latência comuns em redes sem fios. O desenvolvimento e teste dos algoritmos foi realizado num sistema que usa uma topologia de grelha de Manhattan.

Abstract

In recent years, Vehicular Edge Computing (VEC) has emerged as one solution to tackle the challenge of reliable and fast service provisioning to the Vehicular Applications' component of infrastructural computation, in Vehicular Networks, and to accelerate the advent of Smart Vehicles. This paradigm consists of leveraging the ever-increasing computational and storage capabilities provided by the devices at the Network Edge, which are near to the end-users, in complement with the services provided by Cloud Computing. The exponential growth of information to process, the latency-sensitive nature of the vehicular applications and the high mobility of the vehicles constitute considerable real-world challenges that need to be addressed for VEC to perform adequately. Thus, a need arises in devising better-performing methods to manage the Edge and network infrastructure to provide continuous service to vehicular clients.

In the proposed thesis, a new "Rank-Based" heuristic is presented, which takes into consideration previous states of network use to increasingly reach more efficient placements of all required vehicle tasks in any given decision moment. When comparing with the Simulated Annealing heuristic, a reduction of end-to-end latency was achieved. For an image processing application, the observed latency performance of the "Rank-Based" approach against the Simulated Annealing was humble: up to 2.5%. When comparing the number of vehicular tasks successfully serviced, the proposed heuristic was also slightly better, at 3.5%. For a revised safety-oriented application, with less processing requirements, we observed preliminary latency performance gains of up to 5% and a increase in the difference of the serviced tasks, at 10.1%.

Both heuristics rest upon a new Simulation Framework, developed to accommodate different methods to solve the problem of this thesis, and as such, can easily be adapted to hold further improvements of the proposed heuristic or to use entirely different solutions. It can even be used as a platform to the study of other problems, and we hope it can be a valuable contribution for the community who works with vehicular communications and Edge computing. Considerable care was taken to ensure that all values used in the simulations were informed by real-world applications, commercial hardware and common latency factors in wireless networks, in order to provide a high degree of accuracy to the results presented. The development and test of these algorithms was conducted over a system world that uses the Manhattan grid topology.

Acknowledgements

I wish to show my gratitude to my Advisor, Professor Pedro Santos. He made sure all my doubts were clarified and he was very timely with his replies, which I deeply value. Without his insights and experience, this Thesis would not have been as complete as it tries to be.

I also want to thank my friends, who, through these pandemic times we live in, helped me stay positive, happy and never alone.

And last but most definitely not least, I want to thank my mother, Isabel Cipriano. She is my beacon of inspiration. Without her, I would not have had the education and life experiences that truly helped define what I am today.

André Cipriano Sousa

"How lucky I am to have something that makes saying goodbye so hard."

Winnie-the-Pooh (AA Milne)

Contents

List of Figures

List of Tables

Abbreviations

AR	Augmented Reality
B	Bytes
DSRC	Dedicated Short-Range Communication
ERSU	Edge-enabled RSU
FC	Fog Computing
GB	GigaBytes
Gbps	GigaBits per second
GFLOPS	Giga Floating Point Operations Per Second
GUI	Graphical User Interface
HTTP	Hypertext Transfer Protocol
HUD	Heads-Up Display
IoT	Internet of Things
ITS	Intelligent Transportation System
Kbps	Kilobits per second
KKT	Karush-Kuhn-Tucker
Km	Kilomenters
LTE-V	Long Term Evolution-Vehicle
MGA	Mobility-aware Greedy Algorithm
MANET	Mobile Networks
MB	MegaBytes
MCC	Mobile Cloud Computing
MEC	Mobile Edge Computing
OBU	On Board Computational Unit
pmf	Probability Mass Function
POMDP	Partially Observable Markov Decision Process
px	Pixels
QoE	Quality of Experience
QoS	Quality of Service
RAN	Radio Access Network
RSU	Road-Side Unit
SDN	Software-Defined Networking
SRSU	Smart Road Side Units
SMBS	Smart Social-based Message Buffering Scheme
TCP	Transfer Control Protocol
URLLC	Ultra-Reliable Low-Latency Communication
us	Microseconds
V2E	Vehicle-to-Edge
V2I	Vehicle-to-Infrastructure
V2R	Vehicle-to-Road-Side Unit
V2V	Vehicle-to-Vehicle
VANET	Vehicular Networks
VEC	Vehicular Edge Computing
WBSN	Wireless Body Sensor Network

Chapter 1

Introduction

1.1 Context

Vehicular Networks (VANETs) are a significant part of intelligent transportation systems, and their importance will increase as vehicles and pedestrians become ever more connected. According to Gartner Inc. (4), by 2020, there will be a quarter billion connected vehicles on the road, enabling various mobility-related services such as:

- Content-sharing services, relevant to marketing and infotainment purposes, found in applications like autonomous driving or Augmented Reality (AR) techniques;
- Information dissemination applications such as emergency operations about nearby natural calamities or road accidents.

These vehicle services require sophisticated in-vehicle data processing capabilities and large enough storage capacity. Moreover, the on-board services' demands for computational resources grow in a steadfast way. According to Intel (5), a vehicle needs to gather the necessary information about the ever-changing surroundings from its sensors (around 1 GB/s) in order to make safe decisions. There are some applications in which the On-Board Computational Unit (OBU) of the vehicle cannot handle the processing request in a reasonable amount of time to either deliver a service safely or reliably.

Future applications that can benefit from Vehicular Edge Computing can be found in Virtual Reality. For example, in (6), the authors propose a new approach for a Heads-Up Display (HUD) that superimposes an Augmented Reality Traffic Signs Recognition System information on the user's field of vision (on the windscreen for instance). Even in unfavourable weather, this system must perform swiftly in order to provide accurate and potentially life-saving information to the vehicular users.

In (7), the authors explain the development of a high-level architecture for cooperative systems in motorways, where uninterrupted V2V and V2I communication is required. In this example, VEC solutions can ensure timeliness in data processing in order for a driver to share that information between nearby vehicles and in turn, be entirely accepted and perceived as a valuable source of information.

1.2 Motivation

Vehicular Networks are facing challenges to meet the requirements in terms of storage and communication technologies (8). In recent years, Mobile Cloud Computing (MCC) was proposed as a powerful option to fulfil those needs because it could transfer data from vehicles to the cloud server and retrieve the information after being stored and processed. However, the high network latencies of this option, caused by the processing and routing overhead of the network as the cloud server may be located far away geographically, and the increasing mobility and number of the vehicles involved, make it an option that often is not able to produce the necessary outputs within the requirements of the vehicular application (e.g., allowed time frame and bandwidth). A need for alternative communication and processing architectures that ensure low latency and uninterrupted services, as well as being less exposed to the overhead of the communication with remote servers, arose from these challenges. It ultimately leads to a growing interest in exploring the computing power of network nodes closer to the end-client and deployed in the proximity to the vehicular networks, the so-called "edge computing". Vehicular Edge Computing (VEC) is the application of Mobile Edge Computing (MEC) in the particular context of service provisioning to vehicular users. Edge acts as an intermediary between the cloud and vehicles, being able to offer trade-offs in Quality of Service. This technology can take full advantage of the new 5G mobile network which is being deployed, as well as one its features, the Ultra-Reliable Low-Latency Communication (URLLC) service category (9).

1.3 Challenges and Proposed Solution

Vehicular applications, safety related or not, that are demanding in computational power can explore the trade-off between computation capacity and network delay offered by such Edge nodes. However, this option comes with difficulties on the resource management aspect at a macroscopic and planning level throughout the vehicle's path:

- Edge data centres must be selected considering the end-to-end latency to the vehicular user. This means having good estimates of the propagation, transmission and switching delays encountered from one end to the other;
- the availability of resources at the client's vicinity. This means not only the processing speed of Edge nodes and its internal queue capacity, but also the knowledge of the RSUs that are in range of communications for every vehicle;
- the knowledge of the user's route: if it is known *a priori* or not;

- the scaling of the Edge resources to accommodate many concurrent users' requests;
- the deadline of these requests.

These same challenges make up the primary motivations of this thesis and, although some work has been done in recent years, many areas and challenges of Vehicular Edge Computing require a better understanding and new advances are needed in order to prepare the future of road mobility. Section 2.3 further explains the most important technical issues of Vehicular Edge Computing.

The proposed solution is a novel heuristic that manages the resources of the Edge nodes by taking into consideration previous states of network use to increasingly reach more efficient placements of all required vehicle tasks in any given decision moment, a so called "Rank Based" approach. This solution also includes a probability method of using these past results or not, to further diversify the range of possible solutions encountered throughout the process. Chapters 3 and 5 describe in greater detail the specific problem this thesis will try to solve. Chapter 4 shows how the solution and other results were accomplished.

1.4 Contributions

This thesis contributes to the advancement of scientific knowledge in three ways:

1. Proposes a novel Resource Management Heuristic that takes into consideration the vehicular application task's size, the geographical deployment of the Edge nodes and RSUs in the target scenario (a Manhattan Grid), the delays encountered in the network and the computational capacity of the nodes, to ultimately minimise the response time of a task;
2. Development and implementation of a Vehicular Edge node Framework. This framework was developed to accommodate different methods to solve the problem of this thesis, and as such, can easily be adapted to hold further improvements of the proposed heuristic or to use and analyse entirely different solutions;
3. Quantitatively compare the achieved solution's results with an implemented baseline approach, in this case, the Simulated Annealing heuristic.

1.5 Document Structure

This document is structured in Chapters. Each Chapter has a very well defined goal, separated in several topics, or sections, which help structure the thought of the reader.

Chapter 1 presents an introduction to this document. It intends to introduce the thesis in its entirety, giving context and motivations to the problem that is addressed, as well as a brief overview of the proposed solution.

Chapter 2 is comprised of the State of the Art, which introduces essential knowledge that is needed to understand Vehicular Edge Computing better and compiles the bulk of the research done to highlight previous solutions to specific problems which in turn, helped with arriving at a complete solution to the problem at hand. A brief conclusion located at the end of the Chapter collects the most relevant information and puts everything in perspective in preparation for the demonstration of the actual developed work.

Chapter 3 gives a characterisation of the problem this thesis will address. It formally describes the problem, explains a generic System Model as well as it gives an overview of the proposed solution by showcasing the System Architecture.

Chapter 4 presents the resources utilised and the decisions made to implement the system and to develop the heuristics. It focuses on understanding exactly how the goal of this thesis was reached.

Chapter 5 demonstrates how we specified the previously described generic System Model for this work, as well as presents the assumptions taken. It also intends to validate the results obtained by showing the researched values used in the Simulation, and the effort made to make them realistic.

Chapter 6 is where the results obtained by this thesis can be found. It makes a clear distinction, using several ways of comparison, on the performance of the proposed heuristic with the baseline approach. All of the results are considered average values taken from the many runs the finished program executed, in order to have statistical significance.

Chapter 7 puts into a broader perspective the final conclusions one can make from the obtained results, as well as it gives an overview of the whole project and what its development meant to the author and what it means to the scientific community.

Chapter 2

State of the Art

This State of the Art is composed of three major Sections. Section 2.1 gives an overview of the most important concepts and notions the reader needs to grasp in order to understand subsequent Sections. It briefly describes what VANETs, Mobile Cloud Computing, Mobile Edge Computing and Vehicle Edge Computing are. Section 2.2 demonstrates tangible applications of Vehicular Edge Computing in specific scenarios, highlighting the future of the field. Finally, Section 2.3 presents the research work on solutions to the several technical issues of Edge Computing, from resource management problems to data offloading and remote execution of computational-intensive tasks, amongst other issues. Section 2.4 presents the gaps in knowledge made evident in the literary review.

2.1 Vehicular Network Overview

2.1.1 VANET

A Vehicular Ad Hoc Network (VANET) is a network created in an *ad-hoc* manner (when necessary or needed), where individual moving vehicles and other nearby devices communicate over a wireless medium and exchange useful information with one another. In this network, the vehicles and other devices behave as nodes (1). VANET is a crucial component of Intelligent Transportation Systems (ITS) that have gained popularity thanks to advances in wireless technology and the automobile industry.

VANETs are a special kind of Mobile Ad-hoc NETwork (MANETs). They have unique characteristics such as constant mobility, high computational ability requirements, frequent and rapid changes in network topology, and variable network density. However, the key attribute that distinguishes the former from the later is scale. While MANETs involve up to one hundred nodes, are short-lived and are deployed in support of special-purpose operations, VANETs involve millions of vehicles on thousands of kilometres of highways and city streets (10).

Nowadays, VANETs provide infrastructure for a vast range of transportation applications that need to have in consideration the surrounding environment. This includes safety applications, real-time traffic monitoring, real-time driver support, traffic sign warnings, passenger recognition, road barriers or accident detection. This allows drivers to make better decisions. Other applications of these networks are automatic parking fees and highway tolls payment, as well as multimedia communication and access to the Internet (11).

VANETs are comprised of two elements: vehicles and Road-Side Units (RSUs). The vehicles are equipped with communication and processing devices, which allows wireless transportation up to a limited distance and the computation of small amounts of information. RSUs are distributed along the road and are directly connected to the backbone network to expedite network access. These two entities can collect information on their surrounding environment (other vehicles or sensor equipment that are in range) to process and share it to all types of service providers (2). Regarding the way that these entities communicate between themselves, there are two main types: Vehicle-to-Vehicle (V2V) and Vehicle-to-Infrastructure (V2I). This thesis focuses more on the V2I data communication model. Figure 2.1 shows the overarching Communication Infrastructure. It represents the types of communication between vehicles, V2V, and with edge servers which can be connected via Road Side Units (RSU). A more detailed explanation is present in Section 2.1.5.

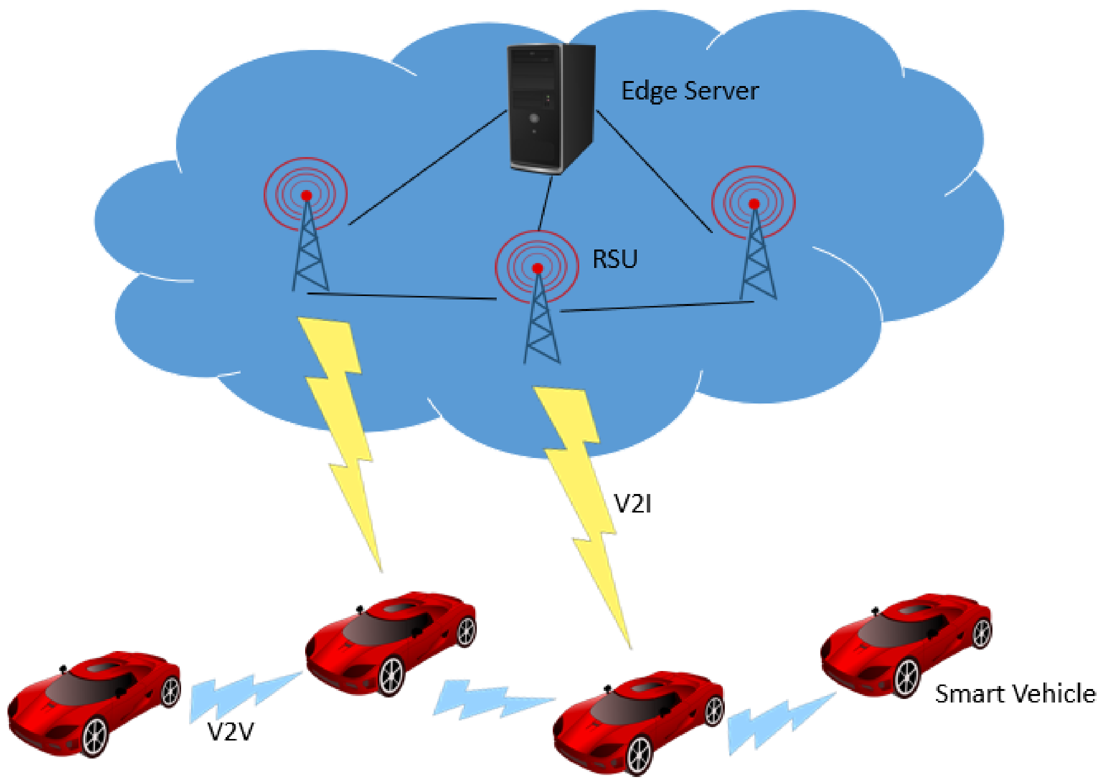


Figure 2.1: V2V and V2I Communication Infrastructure.

2.1.2 Mobile Cloud Computing

The fast development of vehicular networks will continue to facilitate the use of Smart Vehicles, vehicles that are capable of making computations, have local storage and communication facilities and can learn from its environment by making decisions accordingly. The implementation of applications that take advantage of these characteristics calls for vast allocated resources in data storage and processing. Constrained by the limited computation and communication capacities, due to device cost or available bandwidth, vehicles cannot readily satisfy the increasing resource demands, especially for applications that process vast quantities of information within a strict delay requirement (12).

Vehicular Ad-hoc Networks and Smart Vehicles predominantly depend on Cloud Computing capabilities. This term denotes a type of distributed computing, where remote infrastructure (servers) act as a Cloud, or group, in which businesses and users can access applications from anywhere in the world on demand. Cloud computing delivers centralised computation facilities and storage capacity, which can be located far away from the original request location. With Cloud Computing, the stored information can be retrieved at any time; similarly, computationally intensive tasks can be offloaded to more powerful servers, and the result delivered back to the client. This set of operations significantly lessens vehicles' requirements of storage and computational capability in their On-Board Units (13).

Mobile Cloud Computing (MCC) was introduced as a union of Mobile Computing and Mobile Internet technologies with the Cloud Computing mentioned above. Mobile Computing technology is the ability to share resources and transport data between computers or other equipment with wireless capabilities such as cell phones, tablets or Smart Vehicles. The goal of Mobile Computing is to provide valuable, precise and real-time information to clients at any time, at any place. Mobile Internet technology is the use of the Internet on mobile communication. Its goal is to let clients obtain accurate network resources and services from service providers. Mobile Cloud Computing essentially means that any intelligent terminal equipment can obtain services in a wireless environment, provided by Cloud-based services (14).

An important question needs to be addressed when dealing with MCC and Cloud Computing in general, in a Vehicle-to-Infrastructure scenario. That is the unavoidable delay or latency, that can be observed when data is transferred from the vehicles to the cloud server and the retrieval of such information after its storage and processing functions (15). This factor presents an increasing issue in today's world, as already put in perspective in 1.2. To understand the solutions to this pressing issue, an explanation of what Mobile Edge Computing and more specifically Vehicular Edge Computing will be given next.

2.1.3 Mobile Edge Computing

Despite the advantages Mobile Cloud Computing brings to vehicular communication, high transmission latency is a consequence of the cloud being far away from the end-users. Furthermore, the growth of the amount of information exchanged will further increase the load on the backhaul subsection of the network, which extends from the intermediate links between the core or backbone network to the sub-networks at the edges. The bandwidth consumption cannot be disregarded when considering all of the data that is transmitted to the cloud for processing and storage at any given time (16).

Mobile Edge Computing (MEC), commonly interchangeable as Fog Computing (FC), is pictured as a promising paradigm to address these issues. It is a modern architecture where Cloud Computing services, such as storage and processing capacity, are extended to the edge of networks. This edge technology can be applied to mobile and wireless scenarios, by using software and hardware infrastructure situated at the edges, where it can be near the end-users, for example, mobile base stations. These edges can also be an intermediate server which is in close proximity to the user, usually within the Radio Access Network (RAN) (17). It provides seamless integration of multiple application service providers and vendors toward mobile subscribers, enterprises, and other vertical segments, which in turn empowers the end-user with rapid and reliable computing power, energy efficiency, storage capacity, mobility, location, and context awareness support. It is an essential component in the 5G architecture which supports a variety of innovative applications and services where ultra-low latency is required (18).

When directly comparing Cloud Computing with Mobile Edge Computing, several aspects can be analysed:

- **Location** - The services provided by MEC servers are deployed near the end-users, in contrast to the server farms location of traditional Cloud Computing, which can be thousands of kilometres apart. This proximity brings an advantage when analysing and materialising Big Data;

- **Latency and Jitter** - By being geographically closer to the users, MEC latency between communication are physically smaller and less prone to information loss and attenuation. Also, the data movement from user to edge server is isolated from the normal core network data exchange, which reduces network congestion;

- **Storage Capacity** - Cloud Computing has more storage capacity, or at least more potential to be larger because the servers do not have as much space restrictions as edge servers, which often are limited by the deployment location, which need to be near other physical infrastructure;

- **Decision Making** - All of the decisions a MEC platform performs are based on its local environment (distributed), whereas the Cloud has at its disposal information from other detached sources (centralised);

- **Computing Capability** - By being more centralised, which means having space to stack several servers together in one place, the Cloud offers undeniable superior computational power;

- **Context Awareness** - MEC receives information from edge devices within the local access network which provides timely information that can reliably be used by other users within the same network, which may not be possible just using Cloud Computing due to the latencies already discussed;

- **Cost of Development** - A Cloud-based server can cost several thousands of dollars to set up and further utilise, which makes Development a big investment. In contrast, MEC makes use of local resources already in place to deliver its services, which makes it cheaper to develop when needed;
- **Communication** - MEC communication can be considered real-time due to the proximity, as well as having lesser bandwidth constraints;
- **Mobility Support** - MEC platforms can run independently from the core network, as long as they have access to local resources. Cloud Computing depends on the Core Network to operate properly;
- **Device Heterogeneity** - MEC can cope with a wide number of devices running different protocols and operating systems.

In Table 2.1, a comparison of important aspects between Cloud and Edge Computing is summarised.

Aspect	**Cloud Computing**	**Mobile Edge Computing**
Location	Remote location	Close to the user
Latency and Jitter	High	Low
Storage Capacity	Highly Scalable	Limited
Decision Making	Remote location	Close to the user
Computing Capability	Very High	Limited
Context Awareness	Limited	Very High
Cost of Development	High	Low
Communication	Constraints in Bandwidth	Real-Time
Mobility Support	Limited	High
Device Heterogeneity	Limited Supported	Highly Supported

Table 2.1: Comparison between aspects of Cloud Computing and MEC

2.1.4 Vehicle Edge Computing

Vehicular Edge Computing (VEC) has received a significant amount of attention in the past few years due to the realisation of the potential uses that can enhance traffic safety, performance and efficiency and improve travel comfort for the drivers. These new technology applications are the result of an integration of MEC with vehicular networks. The objective of the VEC paradigm is to move communication, computing and caching resources close to vehicular users. VEC has the possibility of being a crucial role in addressing the exponentially increasing needs of edge devices of low delay and maximum available bandwidth, as explained in 2.1.2 and 2.1.3. Different from traditional MEC, the most significant characteristics of VEC are the speed variations in the mobility of vehicles (that range from low speeds on residential areas to high speeds in highways) and the topologies of the roads they pass trough. These conditions lead to frequent and unpredictable changes in the communication channel environment over time (19; 20).

In VEC, similar to what was described to VANETs in 2.1.1, vehicles have access to communication, computation and storage resources such as (but not limited to) RSUs. These devices often act as edge servers and are deployed close to vehicles for gathering, processing, and storing data in a timely fashion. Having edge servers along with the road can ensure communication between the vehicles which allows them, for instance, to interact with one another and inform them in case of an accident or traffic congestion. Due to the limited OBU processing and storage capacity, content requesters rely on the edge servers as platform or middleware service. Vehicles can directly obtain the needed content from caching nodes without accessing the core network or offload their computational-demanding and time-sensitive tasks to the edge servers, which can considerably reduce the response time, improve the efficiency of network bandwidth usage and efficiently alleviate the burden on backhaul networks (1).

The key advantages of Vehicular Edge Computing are now be clearly stated. Several of these advantages could have been already deduced by analysing Table 2.1, as VEC is a subset of MEC. Later on, in Section 2.2, actual applications that make use of these characteristics are demonstrated:

1. **Response Time:** The response time of a request is composed by the time that data needs to be offloaded to the edge servers, the time for being processed in the servers, plus the time to return to the requester. When compared with the Cloud, the edge servers in VEC are closer to the vehicular users. Thus, the service's provision is significantly lower, which is especially beneficial for time-sensitive applications, such as safety applications;

2. **Proximity Services:** Because of the proximity to vehicular users, VEC augments the performance and availability of services that require to be close to them, such as traffic monitoring. This way, the user experience can be more effectively guaranteed while maintaining efficient traffic management;

3. **Context Information:** Edge servers can obtain real-time information related to the behaviour and location of the vehicle, traffic environments, network conditions, among others. With this information, various applications can be improved. One example is that this real-time information can be used to deliver contents to vehicular users based on their interests;

4. **Energy Efficiency:** The increasing prevalence of Smart Electric Vehicles drives a matching increase in applications that maximise their resources. These applications will demand a bigger energy consumption and will put a restraint on electric vehicles with limited energy storage. VEC, by offloading the needed computation for some of the applications, can lower the energy spent by the vehicles;

5. **Bandwidth:** The amount of generated data from vehicles will only grow with the rising popularity of Smart Vehicles. Besides, content requests will also become more diverse. Cloud Computing cannot guarantee the bandwidth requirements for processing such an enormous amount of data computation and content delivery using centralised management in remote

locations. By moving the computation and storage resources of the Cloud to the edge, VEC can alleviate the huge bandwidth stress to the core networks;

6. **Storage:** Caching technologies in VEC platforms enable the access of stored data in time for vehicular users and reduces the storage burden on the remote Cloud.

2.1.5 Typical Edge/Cloud Architecture

This Section puts into perspective the typical VEC architecture, as shown in Figure 2.2 (reproduced from (1)). It is divided into three distinct layers:

- The Cloud layer, composed of centralised servers;
- The Edge layer or Mobile Edge Computing layer, where RSUs often act as edge nodes;
- The User or Smart Vehicle layer.

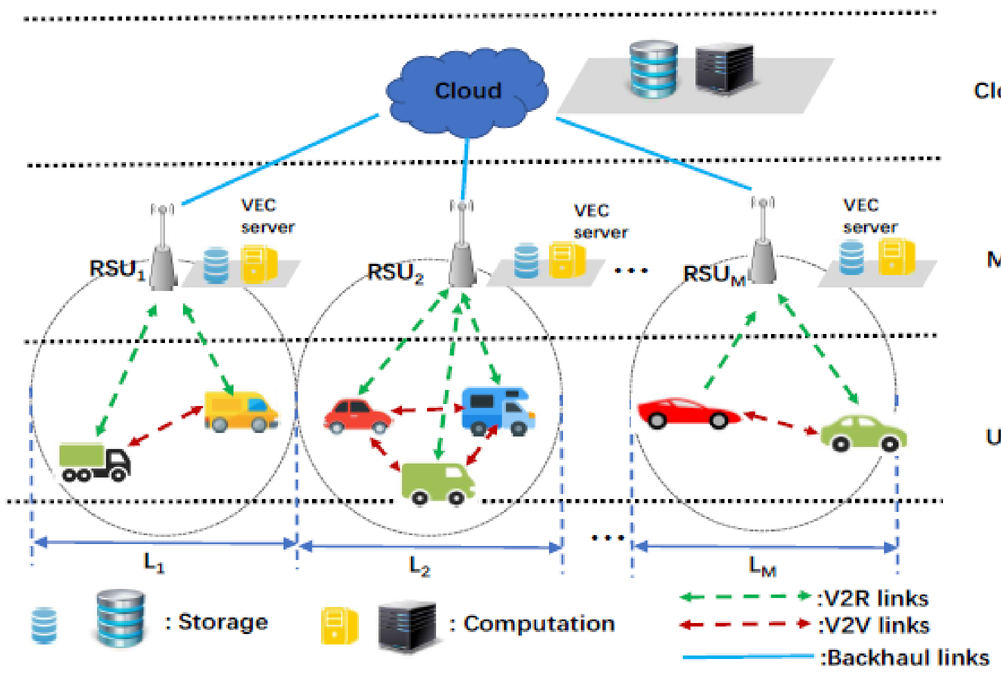


Figure 2.2: Vehicular Edge Computing Architecture (reproduced from (1)).

The Cloud layer has the advantage of being able to permanently store vast amounts of data collected from a broad area that can be used for further analysis (data aggregation and mining), as well as provide complex computational tasks in a short amount of time (analysis optimization and batch processing), compared to the computation power of the edge nodes. The data uploaded from the scattered edge nodes, which is not time-sensitive, and that does not require real-time computation, can be sent to the Cloud instead. It allows the Cloud to have a global view of the covered area and to provide large-scale management and centralized control, which contributes to achieving optimal decisions (21).

The purpose of the Edge layer is to provide low latency and context-aware services, alert for any emergency on the road ahead, cache relevant information for other Users and perform content discovery. The end goal is to improve Quality of Service since its interaction with the end-users is done in real-time (soft real-time). This layer also deals with specific applications which are stringent with its timing needs, such as Augmented Reality or Health and Human behaviour recognition. RSUs or ERSUs (Edge-enabled RSUs which are RSUs that host an Edge node) often act as edge servers in VEC, which are distributed nearby roads in cities or highways. They have higher communication, computation and storage resources compared to vehicles and are responsible for receiving and process time-sensitive information uploaded from the vehicle. They can also forward the information or computational results to the Cloud. As more efficient and diverse computation offloading and caching technologies are developed, RSUs gain increasing value for handling tasks with strict performance requirements.

This layer also provides a communication medium for V2V and V2I. In V2V (or V2R), vehicles interact with those who are in their range of communication. Information can be propagated from vehicle to vehicle until it reaches an edge node. If any vehicle demonstrates abnormal behaviour, for instance in case of abrupt direction changes, speed limit violations or mechanical failures, warning messages are sent to neighbouring vehicles as well as to the edge servers, which stores the speed, direction, current position and other relevant details of that specific vehicle. V2I is liable for the exchange of operational data among vehicles through RSUs, micro base stations, and edge servers over wireless networks (22).

The Edge layer can also incorporate an SDN Controller (Software-Defined Networking), who acts as a network orchestrator, as depicted in Figure 2.3 (reproduced from (2)). Its primary function is to govern the entire behaviour of the network, managing the resources of the edge nodes attached to it (23). Due to the heterogeneous nature of the environment in which this architecture is managed, the entire infrastructure uses different wireless technologies for communication purposes: the communication between vehicles and to the SRSUs (or ERSUs) is done through Dedicated Short-Range Communication (DSRC) or Long Term Evolution-Vehicle (LTE-V). SMBS is a long-range wireless connection like 802.11p, 3G, 4G, LTE, or 5G.

Regarding the User layer, its main actor is the Smart Vehicle. We refer to a Smart Vehicle as a vehicle with the following distinctive attributes. The first one is in its sensing capabilities: is a vehicle equipped with many of the latest embedded sensors such as GPS, Camera, Radar, Lidar, and other devices, that capture both inside and outside environments. The collected information from these sensors can be uploaded to the layer above so it can be stored and utilised as an input to other vehicles that are geographically close or be used directly by service providers. This type of vehicle can exchange information with other vehicles or RSUs using V2V or V2R communication, as previously described. It also has enough intelligence and learning capabilities to allow the possibility of processing part of the tasks required by applications locally instead of transferring all of the computation to the edge servers or the cloud. Additionally, unused storage space can be used to cache pertinent information for sharing with nearby vehicles (24).

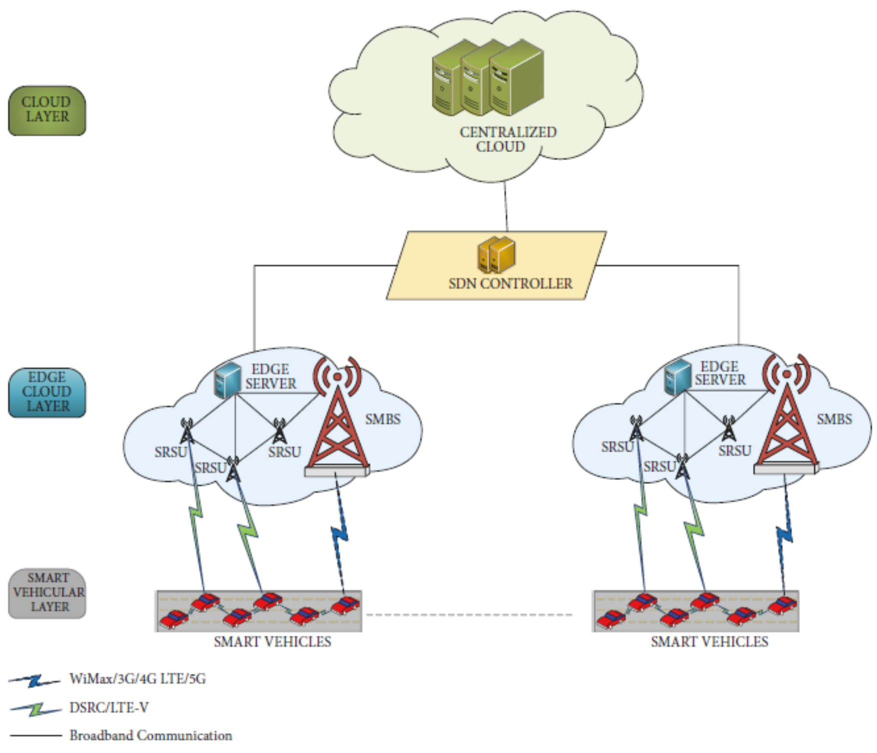


Figure 2.3: Vehicular Edge Computing Architecture with SDN Controller (reproduced from (2)).

2.2 VEC Application Scenarios

Vehicular Edge Computing enables several types of services in vehicular scenarios (25). In this Section, solutions proposed by several authors will be presented, and their relevance discussed. Table 2.3 summarises the examples introduced, where all the applications are mainly categorized by their nature, Safety or Non-Safety (includes automated driving-related services), and by their type, or context of the application. Safety applications keep track of the surrounding environment and alert the driver for any dangerous conditions. Their goal is to minimize accidents by providing timeliness and reliability. Non-Safety applications focus on boosting QoS (Quality of Service) delivered to Vehicular Users.

Table 2.2 shows the Latency Requirement values for some Vehicle-Related Services, which are results of this 2017 study (25). This table helps understand the time requirements involved when making these applications, as well as give reference values to the technical solutions further described in 2.3.

2.2.1 Safety Applications

2.2.1.1 Context-Aware Road Safety

In context-aware vehicular networks, any information that describes the driving situation is called driving context information. The information such as the position, direction, speed, and acceler-

Use Case	Latency Requirement
Emergency Warning	100 ms
Emergency Stop	100 ms
Queue Warning	100 ms
Road Safety Warnings	100 ms
Pedestrian Detection (See-through)	50 ms
Pre-crash Sensing Warning	20 ms
Automated Overtake	10 ms
High-density Platooning	10 ms

Table 2.2: Service Use Case and Latency Requirements.

ation of the vehicle, traffic and weather information, among others are part of the driving context information. In the vehicular network, driving context information always changes dynamically. Context-aware applications in the vehicular network are always aware of the driving situation and road condition and are capable of adapting their operations as accordingly (26).

TOlerant Context-Aware Driver Assistance System (TOCADAS) (27) is a research project aimed to prevent collisions and reduce traffic fatalities. The proposed TOCADAS is aware of uncertain situational information and provides decisions based on the context clues gathered around the vehicle. These clues come from sensor data that is mapped to linguistic variables by fuzzy membership functions, which then it uses to construct a linguistic information system. The control rules are extracted by using association rule mining based on rough sets.

A Collaborative, Context-Aware Application for Inter-Networked Cars is presented in (28). The authors outline possible applications of using the data gathered by sensors with the infrastructure provided by VANETs. They also detail an innovative gas station recommender system that uses the user's gas use information to suggest cheap gas stations located nearby. To be able to deliver trustworthy information, the system has to first determine the number and location of all the gas station within a range of operation, followed by an assessment of the actual range the vehicle has given its current fuel and the distances to the various gas stations. After that, several options can be made to list the results according to price or distance.

In both of these applications, the data gathered by the sensors need to be the most up-to-date as possible, and so, VEC can provide the low latency information transmission that is needed.

2.2.1.2 Health Monitoring System

In developed countries, people often do not have the time or interest to seek out medical help in their day-to-day lives. This modern lifestyle can lead to undiagnosed health issues that affect the well being of their bodies and mind. Health monitoring through VANET is one of the emerging applications that can improve indirect contact between medical professionals and their patients. This can be achieved by monitoring patients using a wireless body sensor network (WBSN), as the authors in (29) propose. This network is composed of temperature sensors, Non-Invasive Blood Pressure devices, microphones or accelerometers sending the personal health information

Nature	Type	References	Section in document
Safety	Context-Aware Road Safety	(26; 27; 28)	2.2.1.1
	Health Monitoring System	(29; 30)	2.2.1.2
	Traffic Control	(31)	2.2.1.3
	Driver Behaviour	(32)	2.2.1.4
	Ultra Low-Latency Service	(33)	2.2.1.5
Non-Safety	Video Streaming	(17; 34; 35)	2.2.2.1
	Augmented Reality	(17; 2; 36)	2.2.2.2
	Vehicle Cooperation	(37)	2.2.2.3
	Platooning	(38; 39)	2.2.2.4

Table 2.3: VEC Applications.

to the OBU of the vehicle, which then relays it to the nearest healthcare centre using VEC. If a life-threatening condition is detected, this can help the ambulance and the medical professional to quickly arrive at the patient's location.

The authors in (30) propose a healthcare monitoring system that is designed to maintain data confidentiality and data integrity, along with authentication in a pervasive health environment. The proposed model aims to deploy a large number of smart ambulance throughout cities that collect information from the WBSNs. These vehicles will be connected using DSRC radios through VANET and are equipped with all the modern digital and analogue equipment, wireless device, computing device, digital map and advanced information processing tools.

2.2.1.3 Traffic Control

Edge servers in VEC gather the current location and speed information from the vehicles located in its communication region. Using this two parameters alongside a database with previous recorded road and traffic conditions can help the servers know about any anomaly in the road, the weather in the area and number of vehicles circulating, which helps in possibly controlling traffic flow and avoid congestion. Carrying out this solution in a distributed fashion can be a challenging endeavour, however. If, for example, there is a distributed algorithm in the cars that depends on all cars knowing the position of all other cars, this overload of information is a communication hazard. Edge can help because of the low-latency that a Cloud computing solution may not be able to provide.

In (31), the authors suggest a congestion avoidance game that is mainly applicable in heavily congested intersections. The congestion problem is modelled as an issue of shared resources competition in a heterogeneous VANET environment.

2.2.1.4 Driver Behaviour

Driving tasks, such as keeping attention to the road and coordinate the appropriate body responses, can be complicated to many drivers to perform them safely and decisively. This complexity is more prevalent in busy city environments, where one wrong decision can cost lives. Using sensors to propose an alert system, which is made up of a collection of functions such as Blind Spot Warning and Traffic Sign recognition, are two of several ways to give drivers a better sense of the physical environment around them. (32) discuses a Context-Aware System, which links drivers to the physical environment to assist and improve their driving behaviour and decisions in critical situations. It proposes a system architecture to handle advanced driver assistance functions and group them all in one alerting system. VEC can improve this system by making sure the environment is rapidly scanned, and all the dangerous variables and possibilities are processed in time.

2.2.1.5 Ultra Low-Latency Service

Due to the fast response time and proximity service capability mentioned in 2.1.4, VEC allows service provisioning with ultra-low latency and high reliability. This is very helpful in scenarios such as autonomous driving because a vehicle that can accurately and timely adapt to rapidly changing environmental conditions has the potential to be safer than a human's reaction time. The authors in (33) refer to several important terms, such as Edge intelligence, which relates to the computing capability at an edge server when provisioning autonomous vehicles. They first consider that the edge server is responsible for the scheduling of content delivery. The edge server first processes the service requests submitted by autonomous vehicles. The requests can be satisfied locally if the requested contents are already cached at the edge of nearby vehicles. Otherwise, the requests are handed over to the remote cloud server.

2.2.2 Non-Safety Applications

2.2.2.1 Video Streaming

Video Streaming is a form of Internet media and file delivery. The files are typically downloaded using Hypertext Transfer Protocol (HTTP) over Transfer Control Protocol (TCP). The efficiency of the transmissions may vary within seconds, because the parameters of the video streaming applications like jitter, buffering, throughput, and transmission delays are negatively affected as a result of variations in the channel conditions or devices entering and leaving the network. TCP may not be able to adapt its response fast enough to the ever-changing conditions in the radio access network, which is a reality in the high mobility vehicular scenario in study, lowering QoS (34).

By merely making use of preemptive caching of popular contents in a region and cooperatively share that information amongst the local edge servers and vehicles, end-users can fetch the requested videos without resorting to the remote cloud, thereby decreasing the delay observed. In (35), the authors investigate the correlation between popularity metrics used in Douyin videos, internationally known as TikTok, which has become one of the most successful short-video platforms on the Web. To maintain its popularity, Douyin has to provide a reliable and fast experience to its growing user base, even in vehicular scenarios like buses or other vehicles designed to carry many passengers. Therefore, understanding the characteristics of Edge provisioning is a worthwhile endeavour.

Also, according to the ETSI White Paper on Mobile Edge Computing (17), end-user Quality of Experience (QoE) and utilization of radio network resources can be improved through intelligent video acceleration. In a plausible scenario, a radio analytics application, which resides in an edge server, provides the video server with an indication on the throughput estimated to be available at the radio area of effect. This information can be used to assist the TCP congestion control decisions on selecting the initial window size, setting the value of the congestion window during the congestion avoidance phase and adjusting the size of the congestion window when the conditions of the channel deteriorate.

2.2.2.2 Augmented Reality

One example of a multimedia application that employs consistently high data rates with a need for low latency computations is Augmented Reality (2). AR is the superimposition of virtual scenes in a device that captures a real-world environment, by utilising computer-generated effects such as images, video or sound. AR can enhance the information conveyed by devices regularly placed in a vehicular dashboard and further increase the traffic awareness of the vehicles, pedestrians or traffic signs within the vicinity of the drivers. Usually, there is a need to update information at a fast rate, depending on how quickly the vehicle moves. VEC can deliver data at this rate, provided the resource managing scheme and latency control used is tailored to this type of application. Additionally, hosting the service on a VEC platform instead of in the Cloud is advantageous since, in AR, information pertaining to the point of interest is highly localised knowledge and is often irrelevant beyond that specific location (17).

In (36), the first Vehicle-to-Edge (V2E) framework to enhance vehicular communication is proposed (EARVE). The authors showcase this framework by realising it in an AR application installed in an IoT device inside the vehicle (HUD or smart rearview mirror), that act as a client. This vehicle, which can capture images that are outside the user's field of view, periodically communicates with an edge server, uploading the captured images and GPS of the vehicle. The server first executes object detection on the camera image with a real-time object detector and then calculates the objects' GPS location. After processing the image to a 2D visualisation, the edge sends the information to the client, which displays it in the AR device. This way, the user is capable of seeing objects hidden by other cars, for example.

2.2.2.3 Vehicle Cooperation

Several concepts are introduced in (37) that relate to the ways Smart Vehicles can use the shared connection between them and to the Edge servers, through the discussed V2V and V2I communication schemes. Network as Service (NaaS) is a way that users with a good Internet connection can share this connection with other users in case of an emergency or to provide a better service to the whole network. Using Storage as a Service (SaaS), Edge servers can provide additional storage capacity to users' applications that require more than what the OBU can manage. Finally, Cooperation as a Service (CaaS) says that vehicles which have unused computational resources, by being stuck in traffic or parking lots, for example, can provide an opportunity to other users to borrow these resources in case more demanding applications need to be executed.

2.2.2.4 Platooning

Within cooperative driving, Platooning is a method that aims to improve the driving experience by automating driving during segments of travel where vehicles are moving in the same direction at a steady speed for an extended time (38). A leader vehicle is chosen to lead the other vehicles while maintaining a pre-determined inter-vehicle space for safety concerns. To do so, delay-sensitive information such as speed, acceleration or braking need to be shared among the platoon members. In (39), the authors propose a platoon-based cooperative approach that avoids backward shockwaves (the sudden breaking of the leader does not negatively affect the breaking of the remaining vehicles) and increases driving stability. They utilise MEC (and by extension VEC) as a way to minimise the transmission delay by moving the computing-intensive and data-intensive virtual machine instances of the global cloud to a local edge server. This example is a direct edge approach to their specific problem, and the results are lower delays in information transmission, as expected.

2.3 Technical Challenges of VEC

Despite the many advantages and applications VEC inherently possesses, there are still persistent technical challenges that need to be addressed to improve the user's experience. The following challenges were written after a reflection of all the papers studied for this document. Thus, their references can be considered to be the same ones as in Table 2.4.

- **High Mobility** - the constant changes in direction and speed are the foremost distinct aspect of Vehicular Networks when compared with the other mobile cases. This particularity makes the connections between the vehicles and the RSUs unstable and subject to disconnections, which deteriorate communication quality. It is also one of the reasons other types of challenges are difficult to handle, as it will be seen shortly;

- **Harsh Channel Environments** - The quality of data transmission is related to the number of obstacles the information passes, and in an urban environment, which are usual in vehicular scenarios, buildings and other vehicles can impose more significant threats than average, particularly in high-density scenarios;

- **Latency** - Due to the high mobility of vehicles, the processing of the information needs to be very fast, lest the temporal utility of the information be diminished. As previously seen, many applications require very low response time to perform normally, let alone reliably. Many of the following challenges, such as Resource Management or Edge Node Placement implicitly try to minimise latency;

- **Resource Management** - The computational and storage resources in the Edge nodes and the OBUs of the vehicles are limited. Thus, it is a challenging endeavour recognise and optimise the available resources of all nodes and vehicles in their vicinity at a given point in time as well as their future states to make sure the maximum number of Applications' requests are timely provided. One of the goals is to try to distribute the load as equally and as much as possible to the edge nodes;

- **Offloaded Execution** - Many times, it is not enough to know how to manage resources. It is also needed a comprehension of which tasks should be sent to which edge node, when this offloading will occur and which ratio of the task can safely be sent to meet the time restraints, assuming a task can be divisible. All of these questions depend on the Application Scenario the solution is intended to act. One important aspect to note is that there is a difference between Offloading and Offloaded Execution: in the former, a vehicle may or may not transfer computations to the edge, having the power to decide; it is an opportunistic execution. In the latter, the vehicular client and a network orchestrator are designed with the intention that they can know beforehand that there is a service needing to be offloaded, and the orchestrator reserves and prepares execution slots in edge nodes which complement the onboard execution in the vehicle;

- **Data Migration** - If the Edge nodes are connected between themselves and coordinated by an SDN controller, for example, it is possible to migrate incomplete processed data to other nodes closer to the current or predicted final position of the vehicle (if the user's route is known *a priori*), allowing a reduction in waiting times and therefore, better responsiveness of the system. This extra capability also adds more questions: what and when to migrate and to which node;

- **Security** - This challenge will not be addressed in this thesis in any capacity since it will be assumed that this is not an issue. However, it is important to note that in a real situation, the communications need to be secure and tamperproof and this topic alone can be the focal point of other works;

- **Edge Node Placement** - Provisioning an infrastructure that is suitable for vehicular applications is a challenge, because, in a real situation, limited funds or available locations for placing RSUs may be unavoidable restrictions;

- **Bandwidth Optimisation** - Available bandwidth can vary depending on location, traffic volume or environmental conditions. In many cases, it can be a deciding restriction in permitting many applications to meet their time constraints. It can be seen as another type of resource management;

- **Energy Consumption** - Although not normally seen in literature, energy constraints can be applied to further impose a realistic view of the interactions the vehicles have with the V2I infrastructure. The more a vehicle utilises remote processing, the more energy is saved that can be used in local processes that are not time-sensitive.

A lot of research and papers were presented over the past ten to fifteen years, detailing solutions to specific vehicular scenarios, system models and architectures. These solutions range from simple heuristics to the use of fuzzy logic or machine learning, to formulating the problems into linear or non-linear programming. This section will focus on explaining some of the issues previously presented, that this thesis will have in consideration, as well as the most recent State-of-the-Art on those topics.

Challenges	Reference	Year	Application Scenario	Type of Solution
Resource Management	(40)	2019	Congested Intersection	Heuristic & Probabilities
	(41)	2018	Manhattan Grid Model	Programming Problem & Heuristic
	(42)	2017	RSUs at intersections	Heuristic
	(43)	2018	Manhattan Grid Model	POMDP
	(44)	2016	RSUs at intersections	Fuzzy Logic
	(45)	2016	RSUs at intersections	Matrix Game
	(46)	2016	RSUs at intersections	Particle Swarm Optimisation
	(47)	2018	RSUs at intersections	Matching Theory
	(48)	2007	RSUs at intersections	Heuristic
Offloaded Execution	(49)	2018	RSUs in a straight road	Support Vector Machine
	(50)	2018	RSUs in a straight road	Programming Problem & Heuristic
	(51)	2019	RSUs in a straight road	Programming Problem
	(52)	2019	RSUs in a straight road	Programming Problem & Heuristic
	(53)	2018	RSUs in a straight road	Heuristic
	(54)	2019	RSUs at intersections	Heuristic
	(55)	2018	RSUs in a straight road	Game Theory & Heuristic
	(56)	2017	RSUs in a straight road	Game Theory & Heuristic
Edge Node Placement	(57)	2018	RSUs at intersections	Programming Problem & Heuristic
	(58)	2012	RSUs at intersections	Programming Problem & Heuristic
Bandwidth Opt.	(59)	2018	RSUs at intersections	Programming Problem

Table 2.4: VEC Technical Challenges and Solutions.

2.3.1 Research Methodology

The research methodology specifically used on this Section of the State-of-the-Art follows the ideas given by the Software Engineering Group of the School of Computer Science and Mathematics, Keele University (60). These guidelines helped in understanding how a well-formulated study selection process should be made in a software engineering context. With this in mind, the methodology was the following:

1. Scientific and peer-reviewed papers and journals were procured in two scientific literature online platforms: Engineering Village and Scopus. The selection process, at first, focused on keeping a record of all literature directly related to VEC (Edge but also Fog related) and

MEC that also mentioned Resource Management, Offloading, Latency Control and other keywords. Nearly 90 records were made;

2. After, a more in-depth look of the contents of the Abstract and Conclusions of each record was made, to understand if the scope of those papers was indeed helpful. This selection brought the number of records to around 50;

3. Finally, based on the year and quality of the remaining papers, a final selection was performed. If they were more than ten years old (unless their relevance was too great to ignore despite the age) or if they did not present any relevant mathematical, logical or heuristically-based approach, they were discarded. The final selection of papers can be seen in Table 2.4. It summarises the thoroughly reviewed works by Challenges' Year, Application Scenario and Type of Solution employed.

An important note needs to be addressed early on as well. Many of the reviewed papers reference two other types of literature that go out of the scope of this thesis, but that are used as a starting point of their publications. The first type relates to publications that relate to any of the earlier concepts such as VANET or MCC but are not focused on Vehicular Scenarios. An explanation of those works will not be made here, and the readers are referred to the cited references if further comprehension is needed. The second and final type refers to widely known works, such as the proof of the Nash Equilibrium proposed by J.F. Nash et al. (61) or NP-complete scheduling problems by J.D. Ullman (62). These works will again be cited when appropriate, and a brief explanation will be given due to their importance.

2.3.2 Resource Management Techniques

Resource management is one of, if not the most pressing technical challenge that received a remarkable amount of attention from the scientific community. One reason why is that when trying to think of solutions to the problem, one quickly realises that many of the other types of challenges can be tackled, in one way or another, at the same time. As many papers will evidence, allocating vehicular tasks in a V2I system is proven to be an NP-hard problem (62). Therefore, a large body of research works has been dedicated to developing resource allocation methods in such systems.

2.3.2.1 Robust Resource Allocation Using Edge Computing for Vehicle to Infrastructure (V2I) Networks

Kovalenko et al. (40) propose a system that dynamically allocates service requests (tasks) among Road Side Units, or, as they call them, Base Stations (BS). They develop an uncertainty-aware resource allocation method for a federated environment of Base Stations (each BS is connected to nearby heterogeneous BS via 5G wireless networks, and each of them is also connected to a central cloud infrastructure). The method assigns arriving requests in a Load Balancer inside the BS that first received a request, that can leverage its computational capabilities as well as those of neighbouring Base Stations, based on a novel probabilistic theory approach to predict the success of task completion within a deadline. This system is robust against uncertain request arrivals, namely, surges of delay-intolerant requests (hazard alerts or lane changing warnings) during peak hours of disasters. Robustness is defined as the degree to which a V2I system can maintain a certain level of performance in the presence of stochastic, or random, events. This Robustness of the V2I system is evaluated according to the number of tasks that can meet their deadlines.

The system model is comprised of the following scenario:

1. Base Stations are located at intersections or on the side of the roads;
2. The roads and the intersection are considered to be congested, which means the BS is oversubscribed and may miss task deadlines;
3. All tasks have an individual deadline that corresponds to the time frame in which tasks should be processed and returned to the vehicle. Based on the specific service required, tasks are separated into two groups, urgent (delay-intolerant, smaller in data size) and non-urgent (delay-tolerant, bigger in data size);
4. A set of tasks is generated by vehicles and sent to the Base Station for processing;
5. Upon arrival to a Base Station, each task is assigned an individual deadline within which it has to be completed. The deadline is the sum of the task arrival time, the average tasks completion time, Base Station slack and the communication delay between Base Station and vehicle (uplink and downlink);
6. Some tasks are projected to miss their deadline in the oversubscription scenario. If such tasks are delay-sensitive, then they are dropped, which is a common practice in oversubscribed real-time systems.

The heuristic can be described as follows: upon arrival of a task to a Base Station, the Load Balancer inside of it immediately allocates the task either to the receiving Base Station or to a neighbouring Base Station. The Load Balancer assigns tasks to the Base Station that offers the highest probability of on-time completion. Every BS maintains two matrices: Task Completion (ETC) time matrix and Estimated Task Transfer (ETT) time matrix. The ETC Matrix contains estimated task completion time distributions for different task types in different Base Stations. Each ETC matrix entry contains a mean and a standard deviation, both obtained from historical execution time information of each task type on each Base Station. The historical execution time information for each task type on each Base Station forms a Normal distribution. The Estimated Task Transfer (ETT) time matrix stores the latency of transferring data to another Base Station. The entries of ETT are considered as Normal distributions and are obtained from historical data transfer times for different task types from one Base Station to a neighbour. Both ETC and ETT matrices are periodically updated on all Base Stations with the help of the central cloud system. Using a convolution between the ETC and ETT matrices (except for the receiving BS where no transfer occurs), a Load Balancer that receives a task of a particular type can calculate the probability of the task meeting its deadline across all Base Stations. Then, it forwards the task to the

BS that had the highest probability. If this probability is zero, the task is dropped. When two Base Stations have the same highest probabilities, the tie can be resolved by considering the standard deviation of two concurrent distributions. Therefore, the preference is given to the distribution with a smaller standard deviation value.

When evaluating the performance of the proposed probabilistic theory and heuristic, the authors chose the most relevant metric as the number of tasks that miss their deadline when specific parameters are changed. These parameters are:

- The increase in oversubscription levels, which means the increase in the number of vehicles and therefore the number of tasks sent to the Base Stations;
- The increase in the percentage of urgent tasks over the total tasks sent;
- Using just one single Base Station and not a federation.

When changing parameters, the authors compare their Best Probability heuristic against three other heuristics:

- Minimum Expected Completion Time (MECT) - this heuristic only utilizes the ETC matrix to calculate the average expected completion time across all the Base Stations and selects the one with the minimum value;
- Maximum Certainty (MC) - calculates the difference between the task's deadline and the average completion time for this task type by only utilizing the ETC matrix. The task is finally assigned to the Base Station that provided the maximum value;
- No Redirection (NR) - this heuristic does not transfer the task to the neighbouring Base Stations. Thus, whenever the arriving task enters the Load Balancer of a specific BS, it has to be allocated to that same one.

The results show that their solutions outperform in every change of parameters. Up to 17% when the oversubscription level increases, up to 20% when the percentage of urgent tasks increases and up to 45% when one single Base Station is used.

2.3.2.2 Optimal Task Offloading and Resource Allocation in Software-Defined Vehicular Edge Computing

S. Choo et al. (41) introduce a software-defined VEC architecture where a controller defines the vehicles' tasks offloading strategy and determines the resource allocation strategy for the edge nodes. They also formulate a problem on the edge node selection and resource allocation to maximise the probability that a task is completed within a predefined deadline. This problem is a well known NP-hard problem (62), and as such, they devise a mobility-aware greedy algorithm (MGA).

A problem is NP-hard if an algorithm for solving it can be translated into one for solving any NP-problem (non-deterministic polynomial-time) problem. NP-hard, therefore, means "at least as hard as any NP-problem", although it might be harder (63).

The devised architecture contains two planes: the control plane, where an SDN controller facilitates the management of all the edge nodes and stores information about them; and the data plane that comprises the RSU's, vehicles and the connections between them. In this plane, it is assumed that a vehicle can communicate with the others if they are in the range of the same RSU. The architecture follows eight steps that start by the vehicles sending their information to the edges. Then the controller, also knowing the current resource usage of each edge node, analyses this information to obtain an optimal strategy that is sent to the pertinent edge nodes and the vehicles in question. This strategy can use both edge nodes and in-vehicle resources.

In the system model, the authors use the concepts of cost per unit of time of using edge nodes. This prevents the vehicles from monopolising the resources of the edge cloud at random. They also make sure to set up restrictions on the edge nodes resources capacity.

The proposed Mobility-Aware Greedy Algorithm ensures a level of fairness to the strategy by allocating resources first to vehicles with fewer alternatives. They introduce the concept of priority, meaning that a higher priority is given to vehicles that have a shorter communication time with the edge cloud. This is because a vehicle with high mobility may pass through many edge nodes, but the time spent in the range of them is small. Conversely, a slow-moving vehicle may have a lot of communication time, even though fewer edge clouds may be in its way.

In the simulation, they use the Manhattan grid mobility model (64), and the results are compared against three cases: the optimal case scenario, all local execution and all edge cloud execution. The defined metric was the success probability of total task success while the vehicles' speed was modified. The results show that the MGA has similar values to the optimal case and considerably better results than the other two cases.

2.3.2.3 Resource Management in Fog-Enhanced Radio Access Network to Support Real-Time Vehicular Services

In (42), J.Li et al. propose two resource management schemes, Fog Resource Reservation (FRR) and Fog Resource Reallocation (FRL). Vehicular real-time services are prioritised over other services so that the services can always be accessed with only one communication hop, to guarantee service continuity while managing to satisfy stringent latency requirements. The focus of this solution is to tackle the issue of service migration when a vehicle leaves the range of an edge node and enters another.

The used architecture does not include an SDN controller and is very similar to the first one presented in 2.1.5. The service migration procedure relies on handovers. If an edge node receives a service Migration Request from another edge node, it decides whether to accept or not based on the strategy that adopted and if it is a High Priority (HP; latency-sensitive) or Low Priority (LP; latency-tolerant) task. If the migration is accepted, it sends an ACK to the requester; otherwise, it sends a NACK. The adopted strategy can be FRR or FRL. FRR states that an edge node has part of its resources reserved for HP tasks. If a service Migration is requested and the node does not have spare High Priority resources, it denies the migration. In the FRL scheme, the difference resides on allowing a certain part of the Low Priority resources to be released and reallocated to serve the needs of the HP tasks, with the drawback of degrading the performance of the LP tasks.

When evaluating the performance of both schemes, simulation data based on the Cologne metropolitan area was used. The authors also took into consideration the differences in the HP and LP requests' rate of arrival throughout a day. The schemes were compared with a benchmark scheme where no resource management is applied. The results show that the proposed schemes can effectively increase one-hop access probability for the HP services, especially when the resource utilisation is more than 90%, up to 30% higher than that of the benchmark. The improvement increases to more than 60% when the resource utilisation increases to 95% at peak hours. However, the FRR and FRL also have a negative impact on the performance of the LP services.

2.3.2.4 Software-Defined Vehicular Networks with Caching and Computing for Delay-Tolerant Data Traffic

M. Li et al. (43) propose a novel joint resource management scheme based on vehicle mobility and system state. The scheme is formulated as a Partially Observable Markov Decision Process (POMDP) to minimise system cost, which consists of both network overhead and execution time of computing tasks.

Extending the Markov Decision Process framework, POMDPs allow for principled decision making under conditions of uncertain sensing, and so they must maintain a probability distribution over a set of possible states, based on a set of observations and their probabilities (65).

The system architecture is the same one as the typical VEC architecture presented in 2.1.5 with an SDN controller. The vehicle mobility model is the Manhattan grid model. The task's computing can be done at the OBU of the vehicle, at the edge nodes or even the Cloud if more processing is required. The results comparison was made against four other existing schemes: Random selection of the node, Data Network only, IoT Network only and Connected Vehicle Network only (V2V). The results show that the proposed scheme has the lowest system cost independently of the time slot allocation amount or the Data Size for computation offloading.

2.3.2.5 On Resource Management in Vehicular Ad Hoc Networks: A Fuzzy Optimization Scheme

Z. Miao et al. (44) propose a Fuzzy Logic-based Resource Management scheme (FLRM) that focuses on storing information about the resources of a network, such as popular news or movies, and determine, based on their popularity (how much they are being requested), a survival time for each stored resource. Motivated by this survival time, the edge nodes can choose to update their resource list or discard them. This scheme tries to tackle the issue of resource sharing within a network of edge nodes with limited storage capacity when imprecise knowledge is prevalent. The popularity of resources is assessed through gathering and analysing data set of request time and download time for each resource. Two important assumptions are made:

- An ideal and error-free environment is considered when transmission occurs;
- Vehicles can complete the communication process with the local server within the idle time.

The Fuzzy Algorithm employs two sets of linguistic variables, request time and download time, that map to "small", "medium" or "large". These are normalised and converted to membership functions. The inference method used is the Mamdani, and the output membership functions can be "Shorter", "Short", "Middle", "Long" or "Longer". Finally, the "deffuzifier" method used is the Center of Gravity. The simulations tracked the performance of the algorithm by measuring the throughput in Kbps, when the arrival rate of requests, the number of stored resources and the time of arrival changed. The comparisons were made against the resource reliability measurement algorithm (RRMA) and the optimisation-based resource management scheduling (RORMS), two schemes by other authors. The results show that the proposed scheme outperformed the other schemes by up to 60% when changing the arrival rate and by 37.7% the average throughput when changing the number of resources and time of arrival.

2.3.2.6 Optimal Resource Sharing in 5G-Enabled Vehicular Networks: A Matrix Game Approach

In (45), R. Yu et al. propose a new paradigm of 5G-enabled vehicular networks, that exploits the high peak data rate for high mobility (1Gb/s) provided by this type of network. The architecture is similar to the ones already mentioned, with edge nodes (or Cloudlets as they refer to them) receiving vehicle requests through associated Base Stations, and an SDN controller orchestrating the global operation. The authors also propose a matrix game theoretical approach manage the resources between Cloudlets to improve resource utilisation. They prove that a Nash Equilibrium (61) can be obtained by a Karush-Kuhn-Tucker (KKT) nonlinear complementary approach. A matrix game (or two-person-zero-sum-game) is a confrontation where n players ($n \geq 2$) whose decisions include multiple factors in a normal and uncooperative context.

Nash equilibrium is a concept of game theory where the optimal outcome of a game is one where no player has an incentive to deviate from his chosen strategy after considering an (constant) opponent's choice, according to (66).

The proposed scheme aims to balance resource utilisation by using resource sharing between nodes. A node can borrow CPU utilisation from other nearby nodes, as long as the link between them can support the data transmission required. This link condition is crucial to determine the success of the solution. Cloudlet resource sharing is based on the conditions of satisfying link quality and cloud resource redundancy. This means that a Cloudlet will prefer to cooperate with the Cloudlets with the most unoccupied resources. To achieve this, a penalty factor, which is an upward curve, is used. The penalty is intended to avoid resource overutilisation in the most sought-after nodes. As a result, the more a Cloudlet is lending its resources to others, the less likely it is to be asked to perform more sharing.

The simulations with resource sharing were compared to another scenario where no resource sharing was used. The results show that, although the average resource utilisation of the proposed solution may not be the best in certain hours of the day, throughout the whole day, it outperforms the no-sharing approach up to almost 30%.

2.3.2.7 A Novel Load Balancing Strategy of Software-Defined Cloud/Fog Networking in the Internet of Vehicles

In (46), X. He et al. propose a novel SDN-based modified constrained particle swarm optimisation algorithm (SDCFN), which enhances the performance of the typical particle swarm method, to decrease the latency of the services provided by the vehicular applications and therefore increase QoS. This latency is defined as the sum of processing latency and communication latency.

Particle Swarm Optimisation is a population-based stochastic optimisation technique that solves a problem by iteratively trying to improve a candidate solution concerning a given measure of quality. (67)

The proposed architecture is similar to the one in Figure 2.3. The same task can be processed in multiple RSUs or Base Stations. The proposed algorithm conjugates graph theory with the centralised capabilities of the SDN controller and with the particle swarm optimisation method, to better balance the load and thus, decrease latency.

The results of the simulation, when compared to an exclusively Cloud or exclusively Edge architecture, show that the proposed SDCFN has lower values of latency for increasing size of tasks. The difference of latency against the Edge architecture is small, but as the size of the tasks increases, the difference gets larger, which proves to be a better alternative.

2.3.2.8 Ultra Reliable, Low Latency Vehicle-to-Infrastructure Wireless Communications with Edge Computing

M.M.K. Tareq et al. (47) introduce a novel algorithm that jointly optimises the distribution of vehicles to Base Stations (RSUs), using concepts from Matching Theory and an iterative process, and the bandwidth allocation, to minimise end-to-end latency. The proposed framework accounts for the latency requirements of the applications running in the vehicles as well as the limited computational and bandwidth resources of the Base Stations. These resources are not the same for every Base Station. Due to this uncertainty, the authors define a probability mass function (pmf) that follows a Gaussian distribution for the computational latency of an arbitrary task.

In economics, Matching Theory is a framework of joining two or more agents to form a mutually beneficial relationship over time (68). The authors allow negotiations for bandwidth between a vehicle and a Base Station when an association is occurring, meaning that the latter offers a specific end-to-end latency guarantee while the former can choose to accept it or chose another Base Station with different terms for the allocation.

The simulations are performed over an area where 40 vehicles and 10 Base Stations are randomly placed. The performance of the proposed algorithm is compared with the current maximum signal-to-interference-plus-noise ratio (max-SINR) and maximum received signal strength indicator (max-RSSI) methods. The results show that the probability of achieving reliability less than 0.8 (in a 0 to 1 scale) is only 30% in the proposed scheme, while this probability for the max-SINR and max-RSSI is 95% and 85%, respectively. It is also revealed that the proposed scheme can guarantee 50 ms end-to-end latency with a probability close to 99%. However, both baseline approaches can only satisfy this latency requirement with probabilities of less than 90%.

2.3.2.9 On Scheduling Vehicle-Roadside Data Access

In 2007, Y. Zhang et al. (48) proposed an early method to address the challenges proposed by the high mobility of vehicles. Although being a dated paper, it still has some relevance to serve the purpose of this thesis, namely in the two unique metrics in which they evaluate their solution, even considering them at the same time:

- The primary goal of the scheduling algorithm is to serve as many requests as possible, which means maximizing the service ratio of requests (the number of requests served before their deadline to the total number of requests) in edge nodes;
- The second metric is what they call of Fresh Data Ratio or Data Quality. Data slowly become obsolete if a vehicle has a new version of important information but fails to upload it before the vehicle moves out of the RSU range, which degrades the data quality for the download service of the overall network.

The solution tries to achieve both high service ratio and good data quality and proves to be effective despite being simple. Its contribution when it was first published helped many other papers achieve a more complex and robust approach.

2.3.3 Offloaded & Remote Execution

This section compiles the studied papers that present relevant solutions to the very much pressing issues of How, What and When to offload information to be processed at the edges. It will be one of the two crucial aspects taken into account in the development of a solution to minimise the overall latency.

2.3.3.1 An Efficient Offloading Algorithm Based on Support Vector Machine for Mobile Edge Computing in Vehicular Networks

In (49), S. Wu et al. propose an offloading algorithm based on Support Vector Machine (SVM) to tackle the issue of fast offload demand in vehicular networks.

The objective of a Support Vector Machine algorithm (supervised learning model) is to find a hyperplane in a N-dimensional space (N refers to the number of features) that distinctly classifies the data points by a clear gap that is as wide as possible (69).

The offloading decision problem is converted into a classification problem so that the SVM, with its machine learning capabilities and with access to a decision tree which uses to train itself, can solve it. The algorithm, running in the vehicles, segments large tasks in smaller sub-tasks using a weight allocation method and then decides if the sub-tasks are allocated locally or in edge nodes. Finally, if some sub-tasks get assigned to be offloaded, the algorithm also decides which edge node that the vehicle passes should compute which sub-task. This solution arose because of a node may not be able to completely compute a requested task in time before the vehicle leaves its range.

The system model scenario is composed of several RSUs located on a straight road. A vehicle can only communicate with an RSU if it is inside its region. The challenges that the proposed solution encounters are how to predict the number of RSUs that should be on the road, which sub-tasks should be offloaded and the size of those sub-tasks. Each node is regarded as a queuing system, assuming a Poisson distribution for the arrival of tasks. The authors predict the number of edge nodes required by making an estimate based on the speed of the vehicle, the processing rate of the servers and the size of the sub-tasks.

In the simulation stage, only one vehicle moving on the road is considered. The performance metrics were the solution conversion time and the average response time (delay between when a task is requested, and the time it is finished and the results returned to the vehicle). A first analysis was made into the accuracy of the SVM, and it was shown that with more training data, the decision accuracy (how the algorithm classifies the tasks to be offloaded or not) improves. It was also concluded that the converge time of the solution increases with the increase in the number of predicted edge nodes needed. This convergence time was compared against the Markov Decision Process (MDP) and FORT (proposed by another author) algorithms and the proposed solution was proven to be faster in reaching a solution. Regarding the average response time, the results show that when compared to a random approach, where the decision to offload or not is stochastic, and a local approach, where no offloading is performed, the proposed scheme presented shorter response times and so, a better performance.

2.3.3.2 Joint Load Balancing and Offloading in Vehicular Edge Computing and Networks

Y. Dai et al. (50) formulate a joint load balancing and offloading problem as a mixed-integer nonlinear programming problem. Because the problem is NP-hard, the authors decide to decouple the problem into two solvable subproblems. Furthermore, they develop a low-complexity algorithm (JSCO) that makes a VEC server selection and optimises the offload ratio and the available computational resources. The performance metric is System Utility, or in other words, the number of computational resources that are used at a given time over the total amount of resources in the network.

The system model is very similar to the one demonstrated in 2.3.3.1 in that the algorithm can choose to offload or not and can determine which percentage of the task to send (both parts are executed in parallel, but the offloaded part incurs a transmission delay). The main difference is that each vehicle can only select one edge node to offload the task to, instead of sending portions of the task to multiple servers along its path. Additionally, in the proposed scenario, five RSUs are randomly placed in a unidirectional road. It simplifies what the previous work did, where the number of RSUs to use had to be determined and was a factor that played into the algorithm.

The performance of the proposed JSCO was compared to three other schemes:

- SO - The Selection Optimisation scheme always selects the VEC server that augments the maximal system utility;
- CO - The Computation Offloading scheme always chooses the nearest VEC server;
- BFS - The Brute Force Scheme corresponds to the optimal global scenario, where all possible cases of server selection, use of computational resources and offload ratio are tested, and the best result used.

The results show that the proposed algorithm significantly exceeds the performance of the CO and SO schemes and is close to the optimal case when the number of vehicles increases. Another simulation tested the algorithms when the number of computational resources diminished. The JSCO maintained a System Utility level of around 88% while the CO scheme dropped to nearly 20%. The load balance test also proved that the solution could make the fairest load allocation across all five RSUs, whereas the SO scheme failed to use two of them, explaining the inferior System Utility value.

2.3.3.3 Delay-Optimal Temporal-Spatial Computation Offloading Schemes for Vehicular Edge Computing Systems

D. Tang et al. (51) evidence the lack of work conducted in determining the point in time when the vehicle chooses to offload. The vehicle may choose to carry the task and reach a new position to obtain shorter transmission and processing delays that offset the extra delay from carrying the task forward. This way, the delayed offloading can reduce the total delay. The authors clearly define the spatial-temporal relation in VEC networks (difficult when considering high mobility) and formulate an energy-constrained delay minimisation problem akin to the work presented in 2.3.3.2. The paper transformed the original problem into task placement (solved using a two-stage decision tree algorithm) and delayed offloading (obtained by dynamic programming) subproblems. An important assumption is that the paper considers the variation of transmission rate according to the distance between the vehicle and the RSU, an aspect many other related works considered constant. It is also taken into account that the vehicle may have reached a new position far from the original RSU when the task has been completed, which causes a longer delay for transmitting directly. With the help of the backhaul network, the total delay may be reduced.

The performance evaluation of the proposed temporal and spatial offloading scheme is compared against a greedy nearest offloading scheme and a more simple spatial (also proposed) offloading scheme, when vehicle speed, input data size and RSU density were increasing. The solution outperforms the other schemes in every scenario, reaching smaller total tasks completion delays, albeit only slightly better than the spatial offloading scheme at the highest values of speed, data size and RSU density. These small improvements beg the question if the added complexity justifies the results.

2.3.3.4 Computation Offloading for Mobile Edge Computing Enabled Vehicular Networks

In (52), J. Wang et al. investigate the vehicular user computation overhead minimisation problem in MEC-enabled vehicular networks by jointly optimising the computation and communication resources' allocation. Since the problem is NP-hard, the authors transform the original problem into two more easily solvable problems and simulate the final solution in the same scenario as 2.3.3.1, very much like the work in (50). The main difference between these two works is that the work of J. Wang et al. focuses on minimising the Computational Overhead (any combination of excess computation time, bandwidth, or other resources that are needed to perform a specific task) of the offloading operation, depending on the task input-data size, the computation intensity or the maximum transmit power.

The results were compared against three benchmark schemes: local computation only, partial offloading with fixed OBU resources, and an SDR-based method proposed by another author. It was demonstrated that the proposed solution dramatically outperforms the other schemes since it takes full advantage of partial offloading.

2.3.3.5 Neighboring vehicle-assisted fast handoff for vehicular fog communications

Y.Bi et al. (53) address the problem of the limited range an RSU has, which causes the vehicle to have to frequently switch its wireless connections from one RSU to another to maintain the connection to the network. The authors propose a cross-layer and neighbouring vehicle-aided fast handoff (CVFH) scheme to enable the always-online connections of the vehicles while moving on roads. The vehicle that uses this scheme is able to get assistance from another vehicle in the range of farther RSUs. The neighbouring vehicle determines the target RSU and gets related information before the other vehicle moves into the area of coverage of the target RSU. This acquiring of information, like the IP address, in advance, can decrease the handoff delay and increase the throughput of the overall service ratio of the network. This handoff delay is defined as the interval from the time a vehicle decides to conduct the handoff to the time it gets the wireless connection from the new RSU.

The system model is based on a straight road with several RSUs placed randomly along the way, which overlap the coverage areas. Seven vehicles, with random speed between 50 and 90 Km/h travel this highway environment and can communicate between them and with the infrastructure around them.

An evaluation of the performance is conducted in terms of handoff delay and average throughput, comparing the proposed solution to the IEEE 802.11 standard, the benchmark. The results show that with the increase of vehicle speed, the handoff delay of both schemes increase, although the solution is nearly 12 ms faster than the benchmark. Additionally, if the rate of packets increases, the handoff delay of both decreases when the rate is 100Kb/s, but still the solution is around 16ms faster than the other scheme. The average throughput follows the same results of the previous two simulations. However, the difference between the two approaches is significantly smaller.

2.3.3.6 A V2X Task Offloading Method Considering Automobiles' Behavior in Urban Area

In (54), H. Matsumoto et al. propose a method that is capable of selecting Edge nodes in a different Application Scenario of what was seen in the Offloading Challenge until this point: the vehicle has the capability of turning at intersections in an urban environment, besides being able to go in a straight line as all other schemes also considered. This new scenario can introduce communications cut-offs earlier than expected due to the turning behaviour, a problem the authors tried to tackle. The proposed solution is a simple algorithm that takes into consideration the turning time of a vehicle knowing its speed to determine the offloading edge node a task must go to meet its delay constraints.

In the simulations, four vehicle behaviours were possible, in an urban environment with many intersections: moving linearly, changing direction after moving straightly for a fixed time, moving straightly after changing direction and finally a combination of all previous behaviours. The metric in question was the average return success of task results, in percentage. The results were compared against two other methods: random selection of the edge node (Random) and a simpler method than the solution where only the speed of the vehicles is considered and not the turning time (referred to as Existed). The results show that the proposed scheme reaches an average return success of 78%, slightly better than the Existed method by 2% and about 15% better than the Random method.

2.3.3.7 A Computation Offloading Algorithm Based on Game Theory for Vehicular Edge Networks

The authors in (55) formulate the computation offloading decision-making problem as a game problem, where the goal of each vehicle is to determine whether to offload a computation task to one edge node or not and make sure all other vehicles achieve a mutually satisfactory solution for their tasks' deadline. For this to occur, the primary performance metric is to minimise the computation overhead, as previous works did so already. The game is referred to as a multi-user computation offloading game, and the basic functioning principle is that the vehicles always compute a strategy profile in which no vehicle can further decrease its computation overhead by changing its strategy, meaning that the game achieves a pure Nash-equilibrium.

In the initialisation of the algorithm, a vehicle chooses its computation decision, which means that a computation task is executed locally. Next, each vehicle computes its set of best response.update. Then, if its time slot has not ended, the vehicle will send a request-update message to one VEC server, which indicates that it wants to update its decision. Otherwise, the vehicle will not change its current decision at the next time slot. A VEC server randomly chooses another vehicle out of the set of vehicles that have sent a request-update message to the server and sends the allow-update message to the second vehicle to update its decision at next time slot. The vehicles that do not receive the allow-update message from the server will not update their decisions and make the same decisions at the next time slot.

The simulations were performed with 30 vehicles, five edge nodes, five wireless channels and with fixed bandwidth, in a straight highway, comparing the results with a random offloading approach and an only local computation approach. The results show that the overhead of the proposed solution is significantly smaller than the alternatives.

2.3.3.8 Optimal Delay Constrained Offloading for Vehicular Edge Computing Networks

K. Zhang et al. (56) propose a hierarchical cloud-based VEC offloading framework, where a backup computing server in the neighbourhood is introduced to make up for the deficit computing resources of the VEC servers when they are not adequate to meet the vehicles' demands. This connection between the Backup Server and the Edge nodes is made through a high bandwidth wired connection, which renders the time delay negligible compared to modern wireless technologies. This connection also has an associated price for using.

Based on this framework, the authors adopt a Stackelberg game-theoretic approach to design an optimal multilevel offloading scheme, which maximises the service ratio of the vehicles and the computing servers. A Stackelberg game is a strategic game, adopted from economics, in which a leader node moves first, and then the follower nodes move sequentially. In this case, VEC servers are the leaders, and the vehicles act as the followers by optimally reacting to the VEC servers' strategies. Each leader sets the price that dictates the cost of using its resource capabilities as well as decide the maximum amount of penalty it can incur by accessing the Backup Server' resources.

Furthermore, to obtain the optimal offloading strategies, an iterative distributed algorithm is presented. Firstly each VEC server starts by randomly selecting both its resource selling price and the number of resources to purchase from the Backup Server. Following the strategies of the VEC servers, the vehicles are selected randomly. Each selected vehicle determines its offloading target server and the amount of the resources needed to offload. Based on the needs of the vehicles, each VEC server firstly adjusts its resource selling price. After, if the demands on the servers still exceed their available resources, then they decide to purchase more resources from the Backup Server. Otherwise, the server decreases its computation capacity until it reaches zero. After, the vehicles respond to the strategy changes of the Edge nodes. Then the servers' cycle of updating their strategies continues until there is no change in strategies compared to the previous iteration.

The metric in the study is the revenue of the service providers. This revenue has to be maximised while making sure the tasks' latency constraints are met. The Application Scenario is a unidirectional straight road, as previous works also took into account. The comparison of the proposed solution, optimal price with Backup Server, was made against an optimal price without Backup Server and a Proportional price without Backup Server as well. The results show that when the number of vehicles circulating increases, the proposed solution is capable of bringing up to 33% more revenue than the alternatives.

2.3.4 Edge Node Placement

Tackling this challenge is not a priority in the development of this thesis. However, if time will allow it, it will be a desirable consideration to improve the final solution even further. It is for this reason than only two papers were considered relevant enough to be in this document, at this point in time.

2.3.4.1 Efficient Placement of Edge Computing Devices for Vehicular Applications in Smart Cities

In (57), G. Premsankar et al. introduces a mixed linear programming formulation that aims to minimise the deployment cost of edge devices for vehicular application in smart cities, while satisfying a target level of network coverage and computational demand given the resources available. Additionally, the paper proposes a simple yet effective heuristic to deploy the edge devices knowing the road traffic conditions in the target area. This heuristic's purpose is to serve as a comparison scheme to the formulation mentioned before.

The architecture considered is the typical case mentioned in 2.1.5. The system model proposed includes an accurate characterisation of complex urban scenarios, including the effect of the environment in the communication between vehicles and edges (the communication occurs over the IEEE 802.11p standard). The target deployment area is modelled as a set of grid cells, each one a possible location for an RSU. Each RSU has a specific transmit power level which puts a limit on the number of cells it can provide. The computation tasks are characterised in terms of CPU cycles, a well-known approach in literature. The computational power of all RSUs is assumed to be the same. Finally, application-specific quality of service is described in terms of both required coverage and processing power.

Regarding the simulation methodology, the proposed mixed linear programming formulation (set of the objective function with associated restraints) is compared against two approaches:

- Uniform - this is the baseline approach, where the RSUs are placed at a fixed distance from each other, in road intersections. According to (70), intersections are the best locations for edge deployment in VANETs;
- Traffic Volume - in this proposed heuristic, the RSUs are placed based on the traffic volume in the target deployment area. If the traffic volume is above a certain threshold, an RSU is deployed at the cell; otherwise, RSUs are deployed further apart of cells that already contain an RSU or other cells which are adjacent to ones also containing an RSU.

The chosen metrics where the percentage of message loss, the number of sent messages that were not received by the recipient; and the percentage of exceeded CPU capacity, the number of messages the RSUs were not able to process due to the unavailability of CPU resources. The lower these values, the better. The authors first analyse the impact of traffic, moderate or heavy, showing that the proposed method outperforms the other two schemes in all situations: less message loss and less exceeded CPU utilisation, even though both metrics increase with heavier traffic. The difference in results from the proposed solution to the Traffic Volume heuristic is bigger, albeit small, which only shows that there is a need for smarter heuristics for placing RSUs, even if they are simple. Despite this, when dealing with thousands of messages exchanged, even a small percentage of improvement is non-negligible. The authors then analyse the metrics when different levels of network coverage and computational demand where chosen. The results show that the proposed method still outperforms the other methods, especially when the requirements go up to 90% of coverage and 100% demand.

2.3.4.2 Optimal Roadside Units Placement in Urban Areas for Vehicular Networks

In (58), B.Aslam et al. propose two methods for placement of a limited number of RSUs in an urban area: an analytical Binary Integer Programming method that implements a branch and bound algorithm to solve the problem; and a Balloon Expansion Heuristic that uses the concept of growing the boundaries of an RSU effective range to dynamically adapts to needs of the vehicles up to a limit. The goal of the schemes is to minimise the reporting time for a target number of RSUs. Reporting time is the time it takes from the occurrence of an event until it is reported by a vehicle to an RSU. The results show that the Balloon Expansion Heuristic is the better approach by being more versatile with the relaxation of the constraints imposed by the system model.

2.3.5 Bandwidth Optimisation

This section only presents one paper since, according to the research methodology, it was the only one that met the criteria of relevance and usefulness. Similarly to the previous section, this challenge will be considered if deemed necessary to strengthen the final solution.

2.3.5.1 Bandwidth Optimal Data/Service Delivery for Connected Vehicles via Edges

In (59), D. Gangadharan et al. propose an optimisation framework that can be used to deliver data (such as update data) or services (such as computation offloading) from the Cloud to the vehicles via edge nodes, in a way that the bandwidth cost objective is optimised. The architecture is the same one as presented in Figure 2.2.

According to the scenario presented, a vehicle first transmits to the Cloud the type of data or service it needs. The vehicle's path is defined as the set of edge nodes it passes on route to its destination (this route is assumed to always be the same). After receiving the request, the Cloud has to decide how to send the information back to the vehicle. The Cloud, given the edge nodes' resource capabilities, needs to take into account:

- The nodes the vehicle passes through, so it may choose if all the data is sent to one edge in the path or be divided into chunks and transmitted over several edges;
- The number of vehicles in the related edges and their speed, which affect the total available bandwidth that is used to send the information or to respond with the appropriate service.

Regarding the available bandwidth, the paper introduces the notion of bandwidth cost based on a well-known bandwidth pricing mechanism (that the authors adopt from another paper). This cost translates to an amount the vehicles have to pay to use the bandwidth of edge nodes: the more bandwidth utilisation, the more the cost. This approach makes the optimisation problem one of minimising this bandwidth use over all the edges.

Based on these assumptions and scenario, the authors propose optimisation constraints that solve the problem as mentioned above, as well as two objective functions: minimisation of max bandwidth utilisation and minimisation of total bandwidth cost. They also demonstrate the usefulness of the bandwidth pricing mechanism as a way to think about the use of edge nodes.

The simulations were conducted by varying the number of edges, and the number/density of vehicles that were requiring data or services. For the first objective function mentioned, the results show that for any number of vehicles, the minimal maximum bandwidth utilisation decreases as the density increases, due to the slower speeds the vehicles have to go. Another result, for the inverse reason as the one before, is that at lower densities, the higher the number of vehicles, the higher the objective function value, which is worse. Finally, the more edges there are in the path of the vehicles, the lesser is the bandwidth utilisation and better the performance. For the second objective function, the same trends as before are observed. Despite this, whereas before it was preferred to minimise at all costs, disregarding the state of usage on several edges, the solutions obtained in this care are more inclined to "even out" the bandwidth use on all nodes.

2.4 Final Remarks

After the thorough review of the State of the Art, some conclusions can be confidently asserted or deduced. The first conclusion is that the vast majority of works only compare their proposed solutions against benchmark approaches or simpler versions of their final solution. This choice tends to skew the results in favour of always being under a positive light. Only a handful of works considers one or two other solutions proposed by other authors. This reality can be explained, in part, due to the absence of standardised metrics or tests that would allow solutions to be compared on a one-to-one basis. This thesis does suffer from the same reality, due to time constraints of making more complex comparison approaches.

Another conclusion that can easily be deduced from the numerical results and the plots all of the papers include is that most of the times, a substantial increase in the complexity of the problem definition and in the equations that describe the behaviour of the system, tends to have diminishing returns of the performance metrics. When compared to simpler versions of the final solution, many of these proposed solutions only are marginally better than their earlier versions. Even though any increase is sought after, the real feasibility for the results they present is questionable.

Finally and most importantly for this thesis work, very few solutions consider the Manhattan grid model. This model, although being simple, represents a more realistic scenario that just a straight road on a highway or only a couple of intersections, because its scale can be easily increased to allow more simulations to be done: unclear offloading cases, more opportunities for different and well-thought task migrations, assignment of tasks' priorities based on streets or blocks of the city or more consideration in placements of the RSUs and Edge nodes. This means that more research work is needed to be done to understand this scenario better. The previous works mostly focused on optimising for one or two variables. Additionally, sometimes the results did not have into consideration the response time of the overall network directly, preferring to address the service-ratio metric, which is slightly different.

Chapter 3

The VEC Management Problem and Proposed Heuristic

This section presents a clear definition of the problem at hand. It includes an explanation of a generic Scenario and of the assumptions made to reasonable simplify some parts of the work. It also includes an overview of the proposed System Architecture and a description of the Heuristics that tackle the issue of the Edge nodes' resource management.

3.1 Scenario Description and VEC System Architecture

A Vehicular Scenario consists of four main entities:

- **Vehicular User** - entity that wishes to execute one or more vehicular applications with Edge Computing support. We define the requests of vehicular applications to the VEC system as being a collection of sub-components, or **tasks**;
- **RSU** - entity that relays information from vehicles to Edges and vice-versa;
- **Edge node** - entity that processes information;
- **The VEC Provision System** - entity responsible for managing the Edge resources to provision the tasks' needs;

A simple explanation of the entities' roles in a typical scenario is as follows:

1. A Vehicular user requests resources from the VEC System for some of its processing service for tasks of one vehicular application that need to be completed within a certain deadline;
2. The VEC System analyses the state of the network and the application's needs and runs a resource assignment procedure to reach a decision, which can be done by using an heuristic approach, for example;

3. Once the resources of the VEC system are assigned to the requested task, the vehicle sends the information to be processed to the Edge node(s) assigned to it; to reach the destination the information must first be sent to one of the RSUs in range of communications, which will then relay to the Edge infrastructure.

4. The information, after reaching the correct Edge(s), is processed;

5. Finally, the processed information is sent back to the vehicle who requested that service, by being forwarded by an RSU in range.

To better understand what each of these steps deals with, a description of a proposed generic Scenario, shown in Figure 3.1, is made. The Scenario Description is essentially the conceptualisation and construction of a scenario and its reasonable assumptions that this thesis had in consideration when devising a solution.

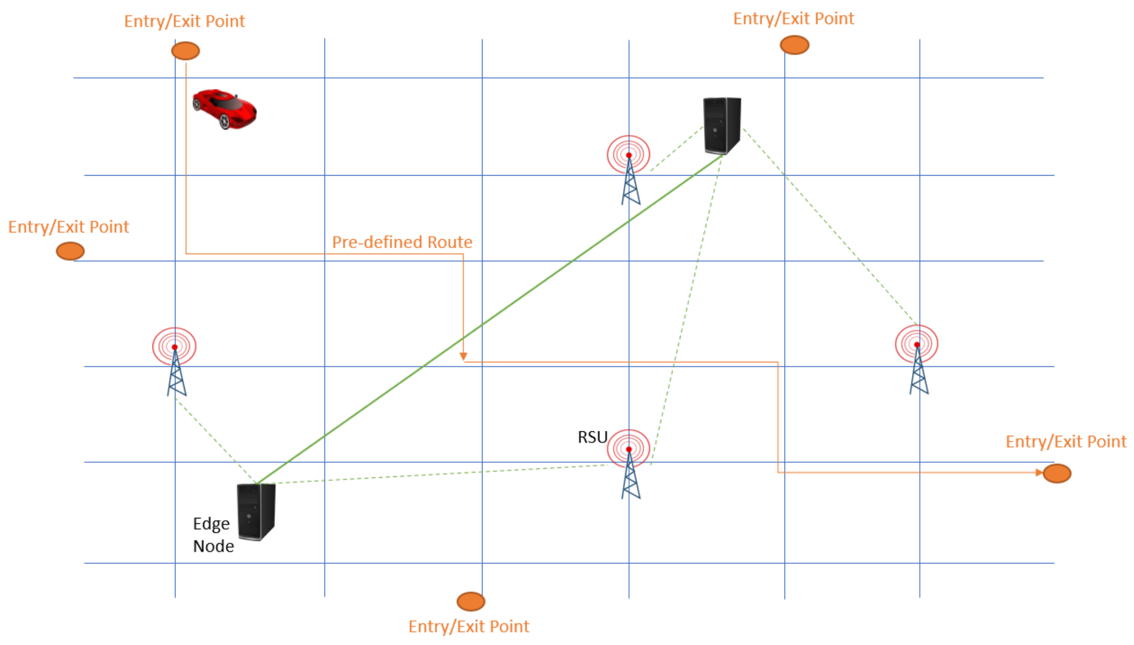


Figure 3.1: A generic Scenario.

This Scenario Description is broken in several parts:

- The vehicles, represented by the small red car, are considered to be **Smart Vehicles**, meaning that they are equipped with an On-Board Unit (OBU). It is expected that up to hundreds of cars can be circulating and requesting service at the same time. They connect to the RSUs using wireless communication protocols, such as IEEE 802.11p or Dedicated Short-Range Communications (DSRC).

- In the Scenario Figure, it can be seen that the **RSUs are only placed near intersections**. In (70), the authors formulate the problem of RSU placement as a Maximum Coverage Problem, and conclude that intersections are the preferred locations to place dissemination points (or in this case, RSUs), due to the high volume of vehicles an RSU can reach over a long period. All RSUs are equal in their nature. Their placement in the Scenario's intersections is randomised. Once set, the locations will be fixed because the exact placement of RSUs was not a point of the study of this thesis. Despite this, the developed framework should modify these locations with ease if needed. In the Figure, they are represented as blue antennas;

- Edge Nodes will be more sparsely placed than the RSUs and do not need to be in intersections. The Edge nodes may or may not have the same computational capability. Their computational capability is measured in Floating Point Operations per Second (FLOPS), as several of the studied papers also use (71). The RSUs are connected to the Edge Nodes through cabled backhaul infrastructure. The location and number of Edge Nodes are not parameters in the study, although it can be easily changed, for the same reasons as the RSUs. The connections are represented by the dotted green lines in the Figure;
- All Edge Nodes are connected through a Backhaul Communications Infrastructure, represented by the green line. Beside normal control purposes, this connection is used to **migrate on-going tasks** to Edge Nodes closer to the vehicles than the original node. The migration can happen due to the mobility of the vehicles, or because the resource management mechanism decided the overall latency is in this way reduced. Although this migration is, in reality, a connection between two Edge nodes, for purposes of simplifying the migration path calculations of the proposed heuristic, we consider that all Edge nodes are connected to a central Edge node, in a star topology, which in turn connects to all other Edges in the vicinity. Under this assumption, **path distance calculations between two Edge nodes is never larger than two hops**. This simplification is reasonable since the transmission and propagation delays in cabled infrastructure are a tiny part of the total delay, as we will see in Section 5.2;
- Packets flowing from vehicles to Edge nodes and vice-versa are subject to a number of **communication and processing delays**, notably: cabled propagation and transmission delays between RSUs and Edges and RSUs; air propagation and transmission delays between RSUs and the vehicles; routing delays if switches are used at any point of the communications process; and the application processing time that occurs at the Edges;
- We do not presume any **road topology**, as the proposed solution should work in any kind of road network: we could be in the busy streets of New York or the middle of a small town in the countryside; the system adapts to different configurations. A simple road topology to illustrate such a case could be the Manhattan Grid model (64). In the Figure, the blue lines are example of streets in which vehicles can move;
- When a vehicle is spawned, a **pre-defined route** may or may not be assigned to it. If it is, the route is randomised within limits and will remain unaltered throughout the vehicle's mobility simulation. The orange arrows in Figure 3.1 represent a possible route. Vehicles enter the scenario following a probabilistic arrival model (e.g., Poisson). The Entry/Exit

Points of vehicles, i.e., the beginning and end of a vehicle's path, are represented in the Figure by orange ovals.

3.2 Problem Description

The problem that this thesis addresses is a straightforward one: **construct a management framework for Edge Computing infrastructure that supports the needs of vehicular applications in a high-density scenario, where the speed of the vehicles is significant.** The vehicular applications are computationally expensive, thus requiring the support of VEC systems. Within this management framework, we will propose a new heuristic for assigning Edge computing and networking resources to the requests of vehicular applications. They correspond to the information needed to be processed in order for a vehicular application to deliver its service.

The goal is to seamlessly provide continuous service to a large number of vehicular users. This can be accomplished by minimising the response time of specific tasks that are required to be processed within a deadline.

This response time depends on the communications component, i.e., how stable is the V2I connection and the associated Round Time Trip, the computation component, i.e., the computational capability of the Edge node and at which speed it can process the required task, and on the vehicle's route and positioning when it requires the Edge services.

An important component that the response time depends on is the vehicular application itself. Many applications could be chosen, as seen in the many types explained in 2.2, but in this work we will focus on a Real-Time Object Detection system. The specific algorithm identified for this purpose is called YOLO (You Only Look Once) (72). This application requires image processing capabilities (frame processing), constant task provisioning, and tight time constraints. We do not actually use it or implement it; it was selected specifically because its developers provide detailed information about its computational needs and performance over a variety of platforms, which is useful when estimating a realistic computation time. That goes hand in hand with this thesis' goal of obtaining realistic values whenever possible.

Regarding the computational capability of the Edge nodes, since we are dealing with an image processing application, we will assume that each Edge has a GPU tasked with accelerating the processing time.

3.3 Overview of the VEC Management System

The architecture of the proposed Management System can be seen in Figure 3.2. For convenience, we describe it according to a Control/Data planes representation. The Data Plane is the set of communication, infrastructure and processes dedicated to application data: is where the application's data exists/travels. The RSUs are only present on this Plane since they are purely passive entities, relaying information as needed. The Control plane is dedicated to all communication,

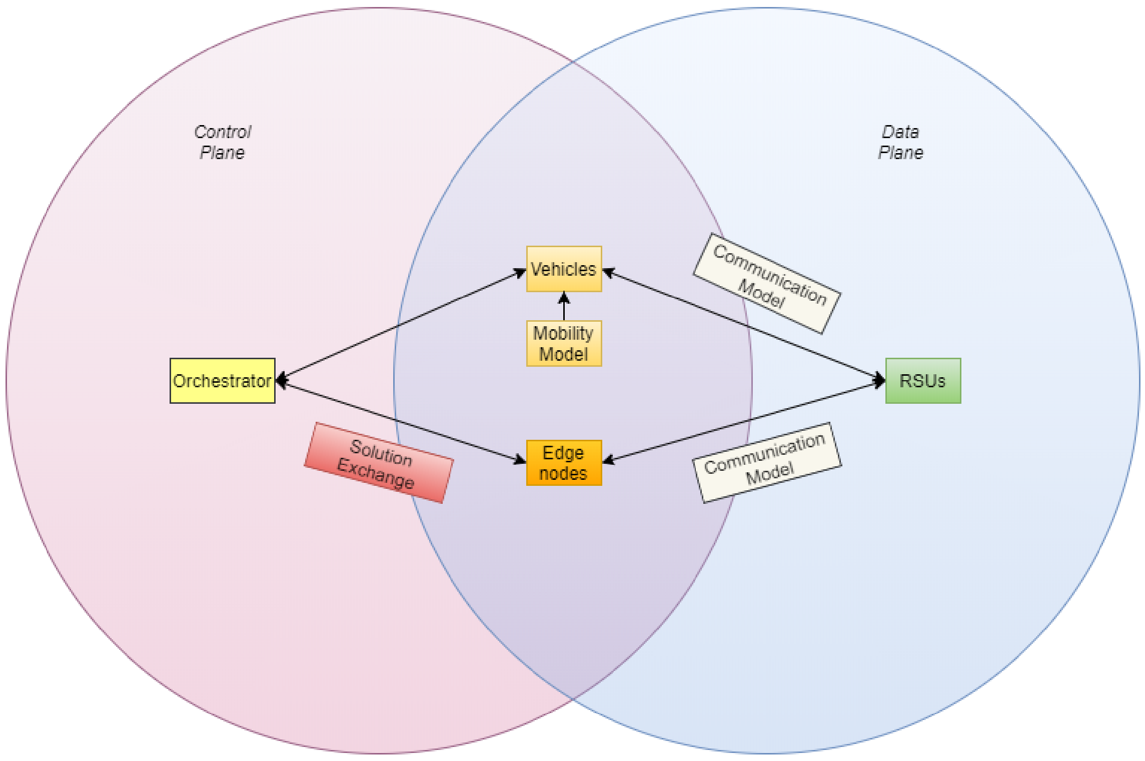


Figure 3.2: Base Management System.

infrastructure and processes required in managing the network. A management entity belonging to the VEC system, the **Orchestrator**, monitors and controls the remaining entities, hence the name. Vehicles and Edge nodes lie in both Planes at the same time since they can communicate with the Orchestrator and the RSUs. The Orchestrator is the most important element of the system because it has direct access to the entirety of the network and what resides on it. We can think of it as a program, located, for example, in a central processing unit with computation capabilities as powerful as the Edge nodes, and which performs several critical functions:

- identifies nearby vehicles' tasks and geographical information;
- manages the Edge nodes' resources dynamically, sending instructions to the Edge nodes on what and when to compute;
- holds information regarding the location and parameters of nearby RSUs and Edge nodes. By centralising this information, still near the edge of the network, effective management of all the resources can be carried out;
- calculates latencies and provides statistics based on accumulated information.

To define the management information shared between Orchestrator and Edge nodes, we define a data structure named "Solution".

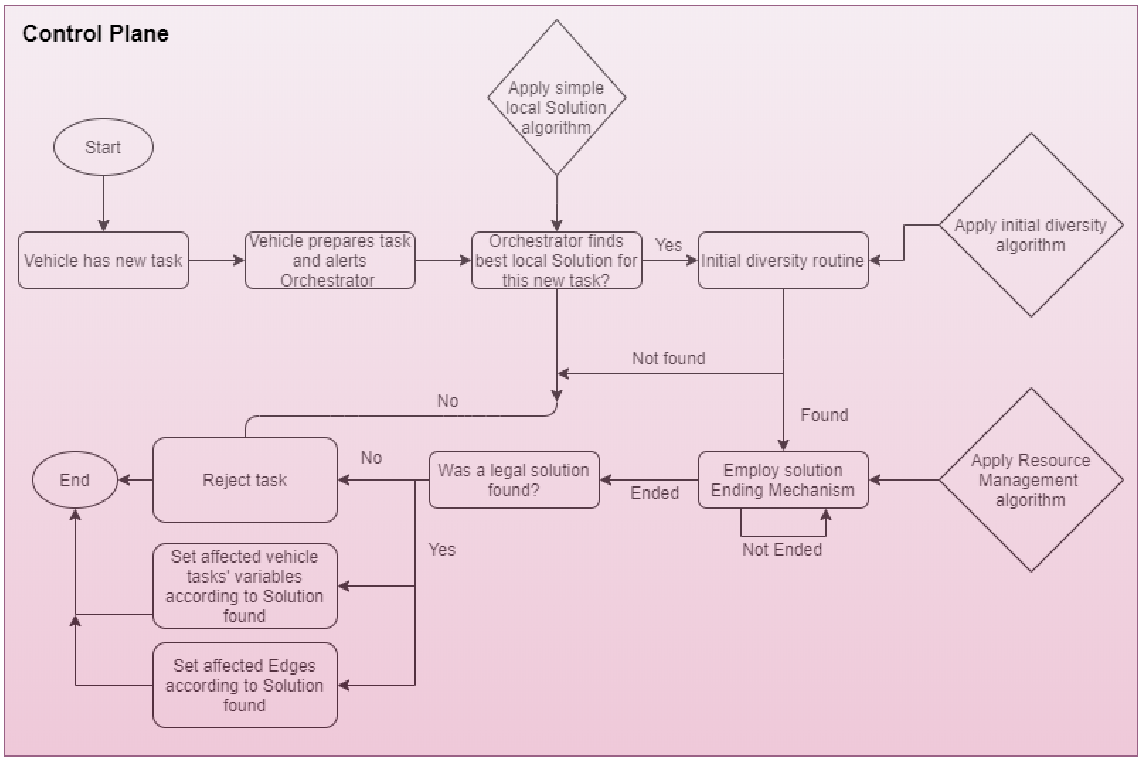

Control Plane
Start
Apply simple local Solution algorithm
Vehicle has new task
Vehicle prepares task and alerts Orchestrator
Orchestrator finds best local Solution for this new task?
Yes
Initial diversity routine
Apply initial diversity algorithm
Not found
No
Found
End
Reject task
No
Was a legal solution found?
Ended
Employ solution Ending Mechanism
Apply Resource Management algorithm
Not Ended
Yes
Set affected vehicle tasks' variables according to Solution found
Set affected Edges according to Solution found

Figure 3.3: Task Information Flow on the Control Plane.

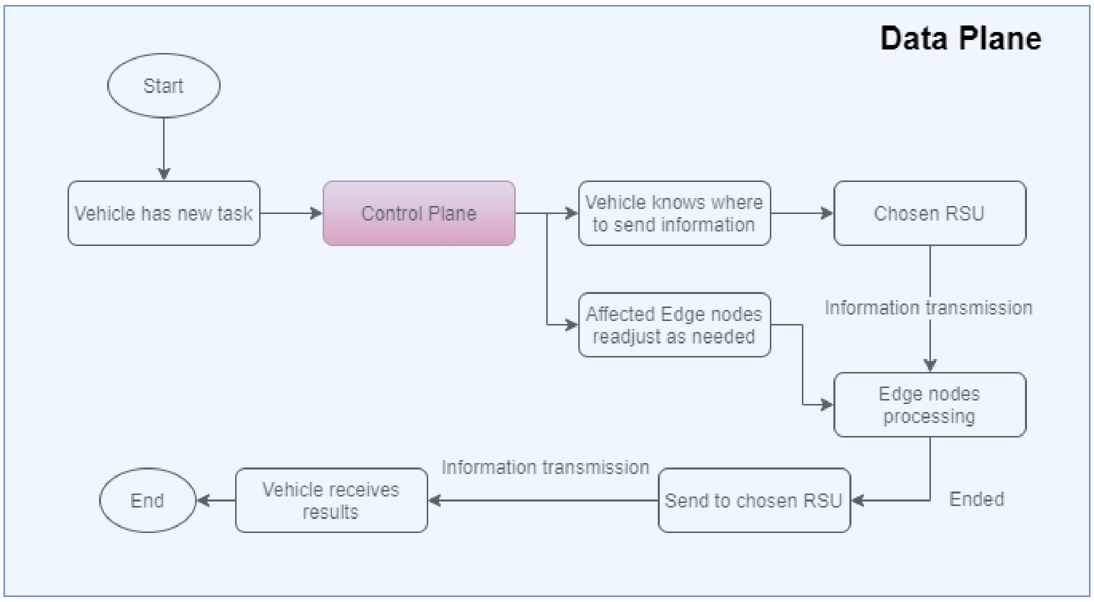


Figure 3.4: Task Information Flow on the Data Plane.

A Solution is a specific assignment of the current processing and pending tasks among the existing Edge nodes' resources. It can be altered as a new reformulation of the assigned resources is computed.

This Solution is very important to the perception of the architecture since it is the way both entities can keep records on the current state of the network. If they do not agree with one another, then the information might be lost or corrupted, and the vehicles could either receive processed tasks that were not meant for them or none at all. Figure 3.5 shows a simple representation of a Solution exchange.

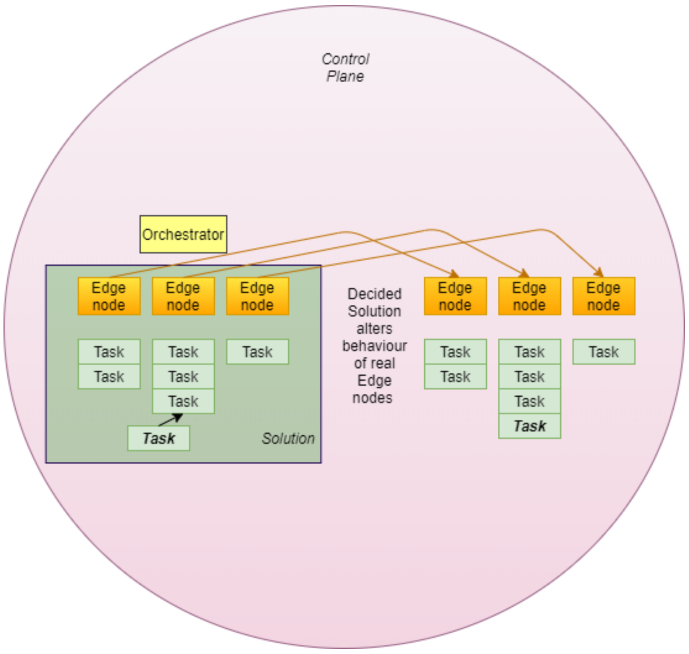


Figure 3.5: A simple representation of an internal network status and Solution exchange.

Edge nodes, RSUs and Vehicles communicate using a Communication Model that simulates real-time wireless and cabled connections and the typical delays associated with them. Finally, the Vehicle's mobility simulation is emulated using a Mobility Model that constantly updates the vehicle's location and speed to be used by other parts of the system.

Figure 3.3 depicts the proposed flow of a task's information in the Control Plane, once a vehicle introduces into the system a new task that needs to be serviced. As we can see, when the Orchestrator is alerted of the existence of a new task, it must find the best local Solution of that task, or more concretely, finding the Edge in which it would be processed the fastest, assuming full availability of all Edges' resources. If the task is unable to be completed even in this way, then it is promptly rejected. The vehicle gets informed about this decision, and the service is denied. If a Solution is found, an initial diversity algorithm is applied in order to find a first **legal Solution, meaning that it has to abide by the space and time constraints** introduced in 3.4: the current use of the chosen Edge must not be over its maximum capacity, and no task inside that Edge is expected to finish over its deadline. Again, if no legal Solution is found, the task is rejected. With the first legal Solution, the Resource Management algorithm must continuously find the best Solution it can until an Ending Mechanism (i.e. a Stop Criteria), a way to stop the heuristic iterations, is triggered. When that happens, we check one final time the legality of the Solution, and if it is valid, the vehicle and the Edge nodes affected by this new Solution get notified with the decision results. It is important to note that the vehicle must receive an RSUs-Edge pair (or trio in this case) of information to know correctly which RSU it must send the task, which Edge should then process, and finally which RSU is the one transmitting back to the vehicle the processed information.

Figure 3.4 depicts the proposed flow of a task's information in the Data Plane. It is the logical continuation of the Control Plane and more straightforward as well. When the vehicle receives the information regarding the RSUs-Edge pair, the information makes its way to to the chosen Edge node's queue. After the processing is finished, the task informs the Edge on which RSU it must send the information to be returned to the vehicle. This may include moving the data from one Edge to another, to reach the specified RSU.

3.4 Formal Definition

In this section, we formalize the problem in a Optimisation Problem fashion, as to better understand the variables involved and constraints applied. The real solution achieved is not a solution to this formal model, but that does not take importance away from having this type of representation well understood.

3.4.1 Variables

- $e \in E$ - Edge node e of set E
- $v \in V$ - vehicle v of set V
- C_v - required computational capacity (data to be processed) needed to provision task of vehicle v
- C_e - total computation capacity (amount of data an Edge can store) of Edge node e
- P_e - processing speed (FLOPS) of node e
- T_{ev} - total time needed to completely execute task of vehicle v when Edge e is chosen to be the executing node; this time is the sum of all the delays present in a normal communication (wireless propagation and transmission delays + routing delay + application processing delay + cabled propagation and transmission delays + medium access delay)
- α_{ev} - binary variable that is 1 when the deadline of task from vehicle v is not met for the Edge e; 0 otherwise

- T_v - local executing time, on OBU; happens when $\sum\limits_{v\in V}\sum\limits_{e\in E}\alpha_{ev}=0$
- T_v^{max} - deadline of task of vehicle v
- β_{ev} - total physical communication time between Edge node e and vehicle v, more than 0 when in range of one another

3.4.2 Objective Function

Minimize total (global) execution time of all vehicle's tasks:

$$min\sum_{v\in V}(\sum_{e\in E}(T_{ev}*\alpha_{ev})+T_v(1-\sum_{e\in E}\alpha_{ev})) \tag{3.1}$$

3.4.3 Restrictions

The total execution time for the chosen Edge must be less than the deadline for that same task:

$$\sum_{e\in E}(T_{ev}*\alpha_{ev})\leq T_v^{max},\forall v\in V \tag{3.2}$$

Only one edge can be chosen per task:

$$\sum_{e\in E}\alpha_{ev}\leq 1,\forall v\in V \tag{3.3}$$

The total capacity used by the tasks that chose the same Edge node must be inferior to the available capacity of that same Edge:

$$\sum_{v\in V}(C_v*\alpha_{ev})\leq C_e,\forall e\in E \tag{3.4}$$

If local execution is chosen, the execution time must be less than the deadline:

$$T_v\leq T_v^{max},\forall v\in V \tag{3.5}$$

When a task is chosen to an specific Edge, the total time taken can not exceed the time of actual communication with that Edge; needed because of mobility:

$$\sum_{e\in E}(T_{ev}*\alpha_{ev})\leq \beta_{ev},\forall v\in V \tag{3.6}$$

The computational capacity in each Edge is more than 0:

$$C_e\geq 0,\forall e\in E \tag{3.7}$$

The processing speed in each Edge is more than 0:

$$P_e\geq 0,\forall e\in E \tag{3.8}$$

The computational capacity in each vehicle is more than 0:

$$C_v \geq 0, \forall v \in V \tag{3.9}$$

3.5 Devised Approaches

This thesis' main contribution lies in the proposal of a new Heuristic which manages the computational resources of the Edge nodes in order to provide the best service to Vehicle Application's demands. This new Heuristic is called **Rank-Based Approach**. As the name may suggest, it uses a scoring system to modify the current rank, or level of success, of Edge nodes, based on past successes or failures. Before we delve more in what it entails, we must first consider a baseline Heuristic, in order to have some form of comparative basis to draw conclusions from. This form of study is very prevalent in academia, as an attentive reader surely noticed in the State of the Art Chapter. The baseline solution considered of this thesis is the Simulated Annealing Heuristic. Algorithms 1 and 2 present algorithmic views to further clarify how both developed Heuristics work.

3.5.1 Baseline Heuristic - Simulated Annealing

In order for the results of the proposed Heuristic to have a greater meaning than just their absolute values, a need for having a comparative approach arose in the design phase of the system. This decided approach was the Simulated Annealing Heuristic, a well-known and straightforward method of obtaining a good compromise between quality of results and the amount of time taken to reach them (73). This is a random-search technique which exploits an analogy between the way in which a metal cools and freezes into a minimum energy crystalline structure (the annealing process) and the search for a minimum in a more general system. This approach works well with small data sets, where the solution region is limited. Simulated Annealing works as follows:

1. First, we prepare an initial Solution and assume this as our starting point. This Solution will be a first, unrefined assignment of the tasks throughout the Edge resources;
2. Then we continuously alter this Solution by using a Random Placement Routine: we randomly select a task and place it either on another position of the same Edge or on a random position in a different Edge;
3. After this location swap, we check to see if this new Solution is a legal one;
4. If the new Solution is legal, we compare it to the previous best-recorded Solution: if it is better, i.e., the sum of the individual latencies of the pending tasks has a smaller value, then we save this Solution as the best one. If it is worse, a random probability determines if we keep this worst Solution to the next iteration, as to introduce diversity in the future encountered Solutions that could allow us to reach a wider Solution range and possibly a better result;

5. This process continues until an endpoint is reached; in the specific case of this Heuristic, each iteration decrements a variable, usually called *temperature*, according to a *cooling rate* until it is below a certain threshold;

6. In the end, after a predetermined number of iterations, we have left the record of the best Solution obtained in the process.

Algorithm 1: Generic Simulated Annealing Algorithm

```
Result: Find new best Solution using task moving until a threshold is reached
Prepare first Solution;
while temperature > threshold do
    new Solution = RandomPlacementRoutine(Solution);
    if new Solution is a legal Solution then
        if new Solution is better than previous Solution then
            best Solution = new Solution;
        end
        else
            if Probability of keeping > random value then
                Keep worst Solution;
            end
        end
    end
    Decrement temperature with cooling rate;
end
```

3.5.2 The Proposed Heuristic - "Rank-Based Approach"

This approach was inspired by works such as (74), (75) and (76), where the authors explain how the use of several forms of Pairwise Comparison to obtain a global view of the accumulated data help when trying to assign "scores" to objects or variables of interest. This scoring allows the system to quickly identify the current situation and recommend the preferred variables to deal with the problems at hand, within a certain level of confidence. To the best of our knowledge, no previous academic work has been made that tried to incorporate these concepts of comparison with stored knowledge into an Edge Resource Management point of view. Kovalenko et al. in (40) do use stored matrices to help their decisions, but they do not adapt the matrices according to the accumulated information and only consider the type of tasks as to estimate timings.

The important novelty/distinction of this Heuristic is that it utilises previous decisions as starting points to determine what the current decision should be. The lower the latency obtained by decisions that took place with similar road and available resources conditions as the one being evaluated now, the stronger the probability that the same decision will be the one employed again.

Placement of tasks on Edges with no previous information to aid this process is a significant drawback, since the time it takes to understand the scenario conditions can impact not only the decision time but how far the Solution can evolve before it must terminate. Let us imagine a new task that needs to be assigned a decision. We know the amount of traffic in the roads nearby, the availability of the Edge resources, the needs of the task and the current state of still processing tasks. These are readily available values stored in the Orchestrator. If we take into account a decision performed when the conditions where approximately the same as the one we now have, we may forgo the need to reevaluate the scenario and lose time on latency calculations that could have been stored in some way for later use. Of course, the more similar the scenarios are, the higher the probability of reusing the decisions. All of the decisions taken are evaluated at the end of a task life cycle, and the results (if the latency was better, the same or worse than the average) help shape the argument of using the same decision more or less often when similar cases appear once more. This analysis of the results is translated into a variable that we call "Merit".

Merit is defined as the quality of a specific decision or Solution. The higher it is, the more likely we will use that Solution again.

When a new Solution is reached, we store it with a base Merit value. Merit is gained and lost proportionately to the Solution's performance: if a Solution performs better than average results, it gains Merit; if it performs worse than the average results, or in the worst case, the task was unable to be serviced, then it loses Merit. There are limits to the amount of gain or loss based on single results, as well as a limit to the maximum Merit Solutions can have, as to introduce a chance of diversity to the final decision: even if a decision is very successful, there is no guarantee it will have the same success in a slightly different scenario than the ones it was used up until that point.

To better comprehend how the information needed to make these decisions is handled, the proposed Rank-Based Approach works as follows:

1. First, we seek to understand if the current scenario is a recurrent one. We analyse the several conditions mentioned above to determine the overall situation we are in and then store if needed, the resulting representation that will allow us to swiftly reuse this information in the future;

2. If the scenario is a new one, or no approximation is close enough to the confidence level desired, we must prepare an initial Solution and assume this as our starting point, in order to then perform a "learning phase", where we try to find the best location of the task in all

Edges, to minimise the global latency. This placement can be chosen in the most varied of ways, for example, the random placement used in the Simulated Annealing. This process is the most time consuming one of the Heuristic since it can be a process with a fixed time to terminate, but in practice, it should seldom be recalculated once a reasonable amount of information is collected;

3. If the scenario is a known one, or approximately so, the algorithm applies the stored Merit and calculates the probability of reusing the previous results. If it decides to use, then the algorithm is finished. If it decides not to use, then it must redo the learning phase explained above;

4. Once finished, the Orchestrator waits for the actual results of the decisions after the processed task has returned to the requesting vehicle and updates the Merit according to the level of success or failure. Further details on how this calculations were implemented are located in 4.3.2.

Algorithm 2: Rank-Based Approach

```
Result: Obtain a Solution based on previous results if possible
Analyse scenario: traffic levels, Edge resources' availability, etc;
Store representation;
if if it considered a new scenario then
    Prepare initial Solution;
    LearningRoutine(Solution);
end
else
    Retrieve Merit of the Solution;
    if Merit > random value then
        Reuse Solution;
    end
    else
        Reset Solution;
        LearningRoutine(Solution);
    end
end
...obtaining results;
if results > average then
    Increase the Solution's Merit;
end
else if results < average then
    Reduce the Solution's Merit;
end
```

3.6 Final Remarks

The main takeaways of this Chapter are:

- a Vehicular Scenario is comprised of Vehicular Users, RSUs, Edge nodes and the VEC Provision System;
- the problem this thesis addresses is to construct a management framework for Edge Computing infrastructure that supports the needs of vehicular applications;
- the Scenario Description presented provided only general details of possible further specifications. Despite this fact, the VEC Provision System and the Heuristics proposed must work in the most complex of cases;
- the VEC management system is divided in the Data and Control Planes;
- a "Solution" is used to share information between Edges and Orchestrator. This concept will be expanded in the next Chapter.

Chapter 4

VEC System Implementation in the Simulation Framework

This Chapter focuses on explaining exactly how the System Model and Management System were implemented in the simulation setup. There is a very tight coupling between System Model, Management System, and Simulation Setup, and therefore all elements are addressed in this Chapter.

Section 4.1 describes more details about the development tools and how they were modified to construct the Edge node framework as close to reality as possible.

Section 4.2 describes the implementation of the System Model and Management System in the simulation setup. Concepts such as "Solution" and the systems behind the entities introduced will be expanded. This implementation is independent of the type of Scenario considered, since it is based in the generic Scenario explained in the previous Chapter. The type of the vehicular applications also does not affect the implementation, as long as specific values can be provided in the setup step of the simulation. As such, only in Chapter 5 will it be discussed details such as the specific System Model or actual values used to bring to reality this thesis' specific Scenario.

Section 4.3 describes the operation and implementation of the Heuristics for resource assignment.

Finally, in Section 4.4, the methods of obtaining the statistics that were used in order to perform the analysis of Chapter 6 will also be detailed.

The System Development Life cycle method used in this thesis was the Spiral Model. This method allowed for an iterative breakdown of work in smaller, more manageable tasks, each with a clear identification of what needed to be done and with the required risk management this project required (which mainly entails the delays occurred during the development when the implementation details were more complicated than expected). Another advantage of using this method is that it allowed for the implementation of more substantial and complete working prototypes of the final solution, as the development advanced (77).

4.1 Simulation Framework

4.1.1 Requirements for Implementation in Simulation Setup

The representation of the entity "Vehicle" in the simulation setup must be able to:

- simulate an application requiring service and prepare tasks accordingly;
- use realistic wireless communication to send and receive information from RSUs;
- have a retransmission system in case there is medium access issues;
- simulate its own mobility and provide geographical data to all entities that request it;

An RSU must be able to:

- use realistic wireless communication to send and receive information from vehicles;
- have a retransmission system in case there is medium access issues;
- use realistic cabled connections to exchange messages with its connected Edge;
- distinguish messages received via cable or wirelessly, and distinguish type of sender entity.

An Edge must be able to:

- process tasks every α seconds, to simulate a GPU executing;
- hold a dynamic queue that is modified based on the current Solution;
- communicate with the Orchestrator to deliver new information or be updated on the current Solution configuration;
- use realistic cabled connections to exchange messages with its connected RSUs;
- use realistic cabled connections to exchange migration or passing-through messages between the other connected Edges.

The Orchestrator, besides of what was already said, must be able to:

- incorporate a "Signal-Based mechanism" to control information sharing in the Control Plane. This aspect will be discussed on its own in Section 4.1.3;
- update its records of the states of all Edge nodes and RSUs as regularly as it needs, to provide the best QoS.

In Figure 4.1, it is presented the proposed System Architecture that provides a global view of the main entities, technologies and the relations between them. This representation is the basis of the main components of the implemented framework. It closely follows the Management System proposed in 3.3.

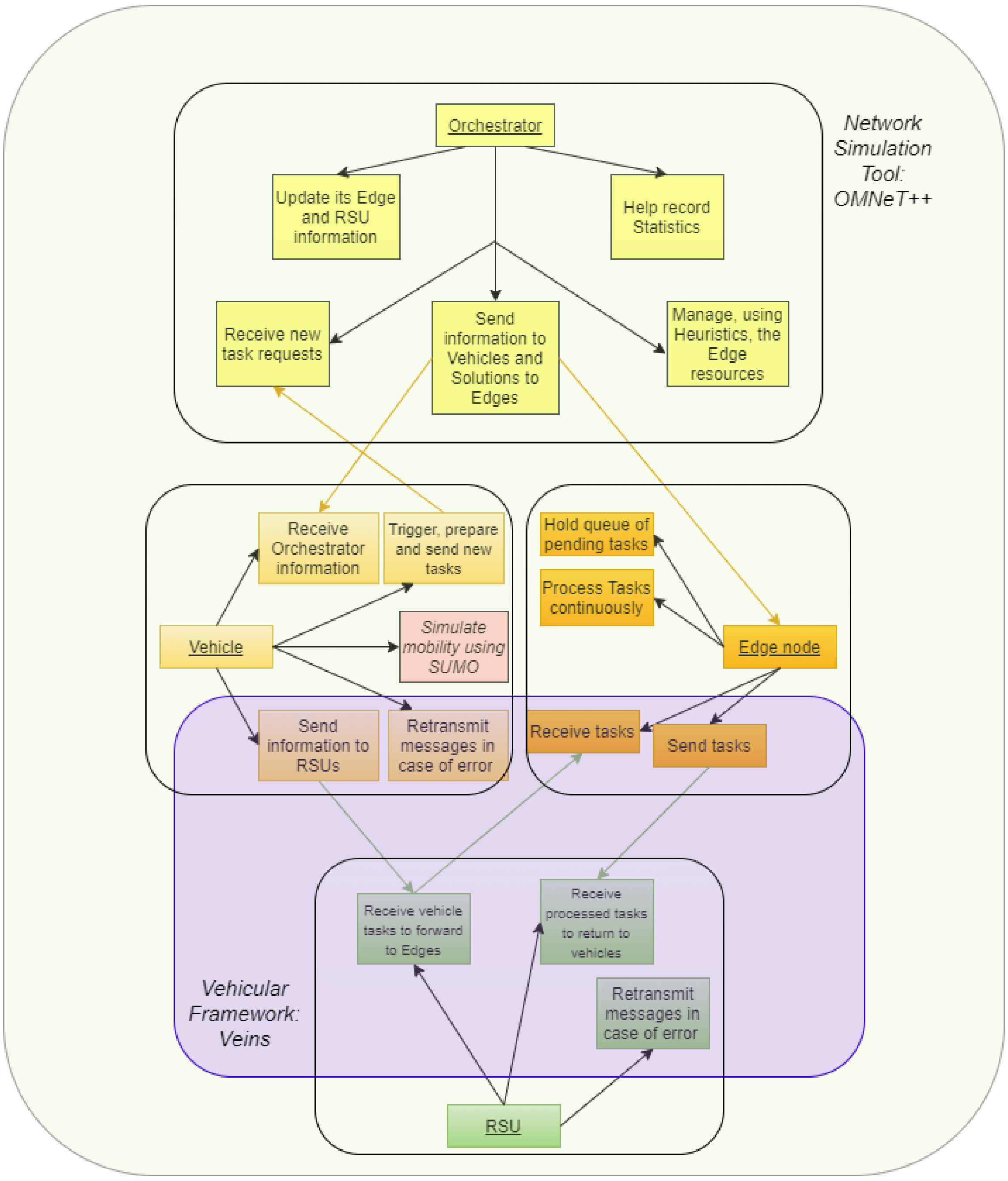

Network Simulation Tool: OMNeT++
Orchestrator
Update its Edge and RSU information
Help record Statistics
Receive new task requests
Send information to Vehicles and Solutions to Edges
Manage, using Heuristics, the Edge resources
Receive Orchestrator information
Trigger, prepare and send new tasks
Vehicle
Simulate mobility using SUMO
Send information to RSUs
Retransmit messages in case of error
Hold queue of pending tasks
Process Tasks continuously
Edge node
Receive tasks
Send tasks
Vehicular Framework: Veins
Receive vehicle tasks to forward to Edges
Receive processed tasks to return to vehicles
Retransmit messages in case of error
RSU

Figure 4.1: The proposed VEC Management System.

4.1.2 Description of Simulation Framework

4.1.2.1 Simulation Tools and Communication-Model

OMNeT++ (78), an event-based network simulator written in the C++ programming language, was the tool used to simulate the vehicular network and the communication model. It allowed the implementation of network topologies made up of modules that simulated Edge nodes, vehicles and RSUs, as well as simplified the creation and modification of cabled connections between them. Each of these modules contained various sub-modules for the control of mobility, communication and application aspects. The tool also simulates the passing of time when information flows in a connection, and it can be modified to introduce, for example, transmission and propagation delays. OMNeT++ is very well documented and has ample online support. Since C++ is a widely used object-oriented language, the vast availability of online resources helped the implementation aspect difficulties to be concentrated on accurately representing the system architecture and less so in the particularities of the language.

Alongside it, we also took advantage of Veins (3), an open-source OMNeT++ framework specialised in realistically emulate wireless vehicular networks. This framework provided several advantages for the development of this thesis work:

- simulated the wireless communications between vehicles and RSUs, taking into consideration real-world phenomena like propagation delay, obstacle collision and wireless medium noise. It has an implementation of the IEEE 802.11p (using a Network Interface Card (NIC)) and IEEE 1609.4 DSRC/WAVE network layers, a Medium Access Control (MAC) layer and a Physical layer, all fully detailing this important component of the RSU-vehicle communication;
- controlled the bidirectional connection of OMNeT with the Mobility Model tool, SUMO, and made it easy to translate the movement of vehicles from the road traffic simulator into the simulation environment, as nodes, using the TraCIMobility module. The protocol for this communication has been standardized as the Traffic Control Interface (TraCI). Both simulators are connected via a TCP socket;
- aggregated information of the physical position of all entities in the simulation, static or not;
- introduced the concept of the Application Sub-Module, a module each entity has, that is responsible for the behaviour of the nodes. It was extensively used in the implementation as evidenced in Section 4.2.
- has a small example program that helps reduce the steep learning curve one experiences when trying to change finer details.

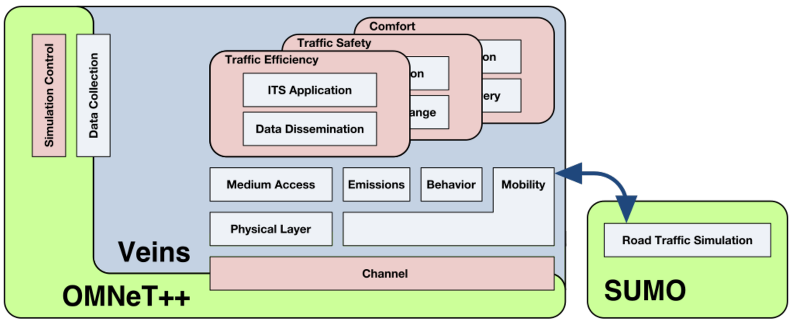


Figure 4.2: Veins Architecture - image taken from the official website (3).

4.1.2.2 Mobility Model

The software used that could perform the required Mobility Model was The Simulation of Urban Mobility, or SUMO (79). It is an open-source traffic simulation package. It was chosen because Veins was established from the ground up to cooperate with this software when simulating the behaviour of the vehicles, and so, it is essential to the development of the solution.

Using a built-in Graphical User Interface (GUI), XML script typing or, more conveniently, by running already developed Python scripts, SUMO allows the creation of XML files that contain road, vehicle and route descriptions. These files can be used by Veins' TraCIMobility module to generate and simulate vehicles as nodes, moving in the Simulation environment according to the specified routes. For this thesis work, the road Grid, or road network, was intuitively made using the GUI, but it could have been taken from a real grid-like section of a city using OpenStreetMap (80); the routes, which contain vehicle information as well, were generated using a Python script called randomTrips. As the name implies, it takes on the network description and creates random routes for vehicles to follow. The script allows the configuration of several parameters such as the type and rate of arrival and spawning point of the vehicles, how many turns a vehicle should make, the maximum allowed speed or how far the start and endpoints of the route should be.

4.1.3 Control Plane Communication - The Signal System

The Signal System is an OMNeT++ feature that allows a module to send a signal to the network system stack. Any other module subscribed to that same signal will receive it, as long as they implement an overloaded version of the native *receiveSignal()* method. A signal is a small unit of information that contains the source path of the signal (the module that sent it), the signal ID (which can be a string variable, for a more natural tracing when declaring) and a variable (string, integer, floating-point...) that stores the information the source wanted to send. Using this form of communication has the added benefit of the processing overhead of the simulation being minimal, which results in an implementation in line with the Plane division of the System Architecture. In this fashion, **we abstract from the overhead of the control communication that the VEC Provision System entails.**

Regarding the implementation details of how the vehicular users and Edges communicate with the Orchestrator, it follows:

- Vehicular Users - the vehicle has declared a signal to the system, and the Orchestrator has subscribed to that same signal name. After the vehicle triggers a new task and prepares it, it emits this signal to all modules. Then, the Orchestrator receives it in its *receiveSignal()* method, extracting the parameters for further use. The variable passed is the vehicle ID, so it can know which vehicle has sent the request and will use this knowledge to send the decision back once it has decided on a Solution. The Orchestrator acts directly upon the task in the vehicle in question, through the capabilities of the OMNeT++ in always having a record of the modules and sub-modules of the simulation. Using clever variable manipulation, one can change the needed tasks' variables the vehicle had declared. Then the Orchestrator sends another signal to the vehicle, which was prepared in the same way as the one used at the beginning of the communication, to permit the vehicle in sending the task to an RSU. Finally, the vehicle receives the signal with his *receiveSignal()* method and proceeds with the execution.
- Edges - the implementation is very similar to the one described above, although this time, only the Edges need to send signals to the Orchestrator. This happens when a new task arrives at an Edge and needs a confirmation acknowledgement. A signal is sent using the same signal name as the one the vehicles possess. The Orchestrator can differentiate if it is a signal sent from vehicles or Edges and execute accordingly by studying the source parameter of the signal. This means that for an Edge signal, it loops through all Edge sub-modules until it finds the one with the same Edge ID in the method's variable parameter. It then accesses the tasks' queue variable inside the Edge directly and searches for the new task to confirm. The operation ends there since the Edge does not need any reply back.

Finally, there is one very important aspect to bring to relevance, as it is one assumption crucial to the understanding of how this work was developed:

While a decision is being found or while information is being shared between one Plane to the other, the Simulated Real Time stops completely.

This means that we do not model the time it takes the Orchestrator to make decisions and for those to be reflected in the affected Edges and vehicles, as well as we do not consider the time it takes for a new task to signal the Orchestrator and send the required pre-data needed for the Orchestrator to make a decision. The Simulation Tool used only simulates communications delay and not code related iterations, however long those may take.

4.2 Module Implementation in the Simulator

In this Section, we are going to understand how the system entities (Vehicle, RSU, Edge, and Orchestrator) are implemented: how they communicate with one another, how they store and handle information internally and how several real-time mechanisms such as retransmissions and latency estimations were incorporated among them.

4.2.1 Vehicles

A vehicle is an entity that resides in both Control and Data Plane. It is considered an active entity since it has the power to kick-start system operations by itself (requesting a task, for example), or in other words, it does more than just react to what other entities do when interaction occurs. A vehicle has several Sub-Modules, each one responsible for certain tasks:

- **Mobility Sub-Module** - know its position, speed, and route at all times, and update the simulation accordingly;
- **Application Sub-Module** - prepare the chosen application's tasks, signal the Orchestrator in order to decide what happens to them; simulate the OBU's processing capabilities;
- **Communications Sub-Module** - send task to the assigned RSU via wireless communication and be ready to receive the processed tasks via the same medium;

In the following Sections, we explain how the vehicle accomplishes these responsibilities.

4.2.1.1 Mobility and Task Triggering

Veins' TraCIMobility Sub-Module handles the mobility component of the vehicle. It requires a route XML file generated from SUMO and the actual SUMO simulator running the file to be able to represent the vehicles in the simulation as "nodes." Every time SUMO changes the position of any simulated vehicle, the corresponding node updates as well. This Sub-Module was modified to handle task triggering: from the moment a node is placed on the simulation and starts its route, a dedicated method schedules the first task request to happen T_{start} seconds later. After this time has passed, the Sub-Module handles this self message by changing a variable to True, which signals the Vehicle Application to prepare a new task. It then schedules another task and repeats this process as long as tasks need to be sent. A parameter can change this, called N_{tasks}. This process represents a periodic requesting of tasks that an application may require.

4.2.1.2 Task Preparation and Requesting

This main responsibility falls over the Application Sub-Module. Every time the position of the vehicle is updated, a method is called to check if a new task was triggered. If it was, it must set up a new task to send a message to an RSU. This preparation involves: setting the deadline of the task, T_d, needed for the Orchestrator to make decisions; setting the amount of computation required to be processed, C_v, used in the Edge nodes; save the ID of the vehicle, important to trace the task in the network; save the simulation instant of the request of the vehicle, needed when making decisions, as well as for calculating to check if the task's deadline was met; set the message as a vehicle-RSU kind, as an internal check when transmitting messages.

Then, before sending the message, two important pieces of information need to be saved in the task: to which RSU it must be sent to and to which Edge node it is headed. This requires communicating with the Orchestrator through the Control Plane, and the way to perform this operation is by using the Signal System explained in 4.1.3. The vehicle sends a signal, which the Orchestrator is ready to receive, and then waits for a signal back as a confirmation to proceed. Once this confirmation arrives, the application layer can send the task by **broadcasting** the message (called Air Frames by Veins) to all RSUs in range expecting of reaching the chosen RSU, since a Solution was found that satisfies it; or it can discard the task if no Solution was found that meets the deadline. However, if it is sent, the transmission is accomplished using a Veins built-in method that handles the propagation of the information using real Network and MAC and physical layer protocols, inside the Communications Sub-Module. After a careful study of the Veins features, it was concluded that it does not model the Wireless, or Air, Transmission Delay usually found in this type of communication. So, part of this thesis work was to model this and other such communication particularities. This value, along other such values that will be referred in this Section 4.2 are explained in Section 5.2.

One other note is that vehicles possess computational capabilities of their own, within their OBU. This OBU is normally much less powerful than the GPUs on the Edges, as further details in 5.2.1.10 demonstrate. So, although the Orchestrator may decide that the Edge resources are not capable of providing the task with enough service to meet its deadline, it can, if the deadline is generous and the task low on computational demand, decide to shift the task to be processed on the OBU. This interaction and processing was implemented as well, despite the vehicular application used in this thesis, 5.2.1.1, being computational demanding.

4.2.1.3 Receiving a Processed Task

After a task is sent, the vehicle waits for the reception of the processed data back. Due to the broadcasting nature of wireless communications, the vehicle ID previously saved in the task is now used to know if the incoming RSU message is destined for this vehicle or not. If it is, we save the time of arrival, and the simulation of that task stops there and then. Then we focus on comparing the arrival time with the instant of request to see if the task did or did not meet the deadline, in order to update statistics for later analysis. Finally, we discard the task. If this task is only one of the several tasks the vehicular application requires, the explained process will repeat N_{tasks} times. One design consideration which will be expanded in Section 5.1, is the F_{limit} variable. This variable indicates how many tasks, of the N_{tasks} an application can request, the system is allowed to fail to complete on time. If this value is reached at any point of the simulation, the vehicular user which requested those N_{tasks} tasks finishes its operation, the remaining operating tasks get discarded once they arrive back to the user, and the application gets marked as failed/refused by the statistical tools.

4.2.1.4 Medium Access Issues and Message Retransmission

A phenomenon that Veins simulates is the presence of noise in wireless channels. This is perceptible when two or more broadcasted messages arrive in an RSU at almost the same time (200 us apart or less, according to measurements taken): the first message is recognised, but the following are discarded as noise by the MAC layer at the receptor. This happens because more than one message is trying to access the same medium of communication within a too-short time frame. To prevent important messages, such as tasks, to be lost, a retransmission mechanism had to be re-implemented in the vehicles' application layer, illustrated in Figure 4.3. Since we are using IEEE 802.11p broadcast mode, given that Veins does not allow unicast, this mechanism was a necessity for the correct functioning of the system. When the first message with the task information is sent to the desired RSU, the same message is scheduled to be transmitted again, $T_{retransmit}$ seconds later. Then the vehicle waits to receive an ACK message from the desired RSU. If it fails to receive, it will automatically resend after the defined time passed, repeating the process as many times as needed until the vehicle receives the ACK or the task deadline has been reached. To prevent cases in which repeated failures occur due to an RSU receiving messages from the same vehicles every time, and canceling other out continuously when retransmissions are scheduled, there is a small random factor, $T_{randomMA}$ that introduces a slight variation in scheduling patterns. With the proper variation, the chance of a successful first transmission, that does not significantly compromise the success of a task being done on time, rises to 94%. The exact values used for both variables here introduced are clarified in the Values Section of this document, 5.2.

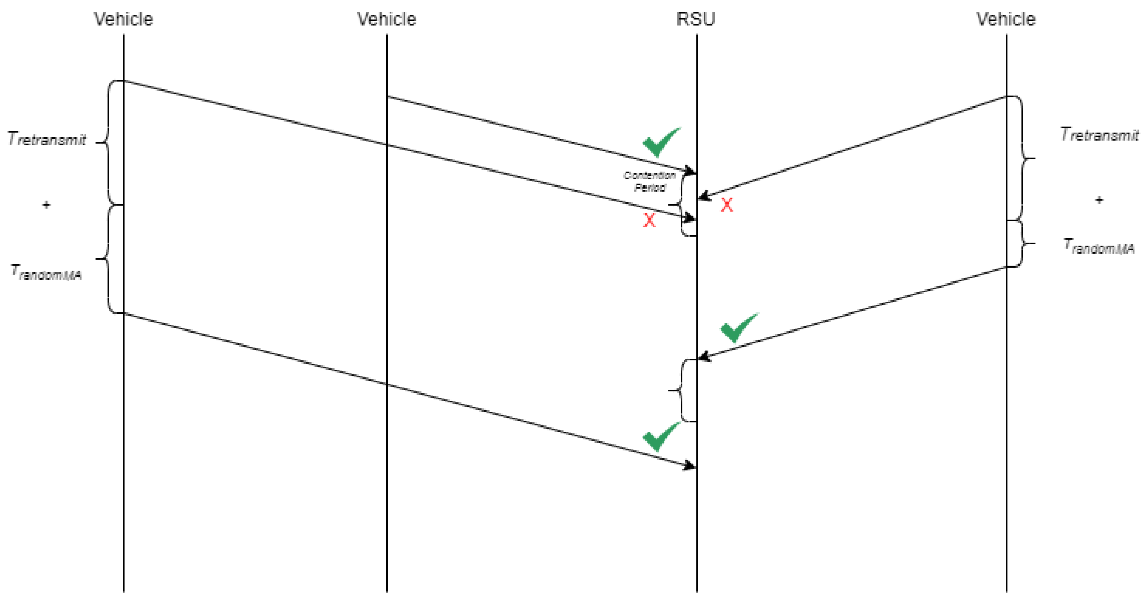


Figure 4.3: Retransmission Mechanism.

4.2.2 Edges

An Edge is an entity that resides in the Control and Data Planes and it is considered an active entity. It has the following responsibilities based on each of its two Sub-Modules:

- **Mobility Sub-Module** - know its geographical location;
- **Application Sub-Module**
  - Receive and send messages from and to RSUs;
  - Confirm and hold all of the designated tasks in an internal queue;
  - Process the message on the head of the queue;
  - Migrate tasks as needed;

We will now explain how the main Edge functions are implemented.

4.2.2.1 Message Handling and Task Migration

Edge nodes primarily receive messages/tasks via the cabled connections to their RSUs. When an Edge receives a task, it checks the Edge ID that was assigned in the vehicle by the Orchestrator, and if it is the correct Edge, the delivery operation stops and the message is forwarded to the storing system inside the Edge. If the Edge that first receives the task is not the final destination, a Migration mechanism handles its delivery. This mechanism consists of sending the task to one of the up to two Edges connected by cable. This cabled connection is identical to the one used between RSUs and Edges. The receiving Edge, if not the destination Edge, proceeds to send it to the other connected Edge, the one that did not send it the message in the first place, if available. To better illustrate this procedure, Figure 4.4 shows an example of Migration in a simple network.

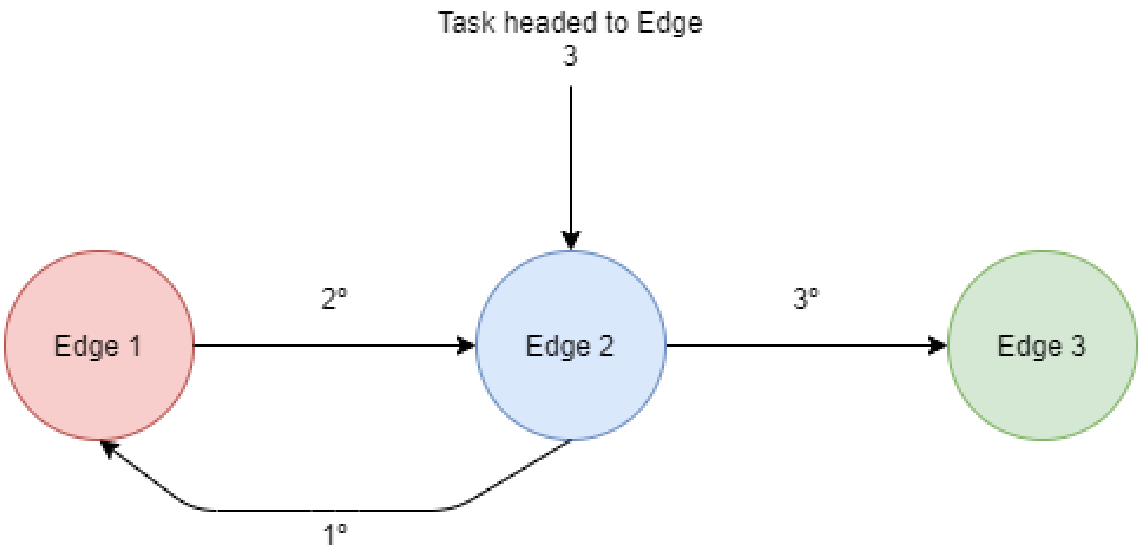


Figure 4.4: Migration Mechanism.

As we can see, Edge 2 received a task, but the contents were destined to Edge 3. Then we send it to Edge 1, which in turn, sends it back to Edge 2 since it has no more connected Edges. Then, the only alternative is to send it to Edge 3 because it is not the Edge the message came from, and the destination is reached. We could have sent directly to the right Edge, but that may not be so straightforward in a more complex network. The goal of this thesis did not require to implement a sophisticated pathfinding algorithm, since one assumption in that all Edges are connected to a central Edge, as in a Star topology network. The propagation delays simulated by such mechanism are so small compared to the overall latency that this reasoning has no significant effect on the results. So, we send it to the Edge that did not send us the message in the first place. This simple pathfinding works well if each Edge has only up to two other connected.

4.2.2.2 Task Storing and Processing

In each Edge, there is a processing queue, which stores received tasks according to the current Solution and processes the one at the head of the queue. This processing, consisting of removing the processing speed value, P_e, from the task's required computation, C_v, repeats every α seconds. An example with real values is given in Section 5.2. The processing loop is simulated as a continuous scheduling of a loop message inside the Edge, using the *scheduleAt* method native to OMNeT++, similar to what is used in the Retransmission mechanism. The main difference is that this loop only ends at the end of the simulation.

Assumption: A task being processed cannot be moved to another Edge in the decision process of the Orchestrator. This prevents data corruption.

Once an Edge receives a task meant for it, it must signal the Orchestrator that a new task has arrived, in order to have it confirmed. To confirm a task means that the Orchestrator now knows that the task arrived where it was supposed to, and is an important factor in the decision-making process since no ongoing processing task, the one that is at the head of the queue, can be migrated to another Edge by virtue a new decision. A task can only be processed if it is confirmed. This confirmation uses the Signal mechanism prevalent in the Control Plane, explained in 4.1.3. After the confirmation, the task is placed in the correct position of the queue, according to the current Solution expressed in that Edge. Figure 4.5 exemplifies this placement. The task then waits for the queue to reach its turn in order to be processed. Meanwhile, if a confirmed task is migrated to another Edge due to a new Solution, then that task does not need to be reconfirmed since it is assumed to be in the Edge part of the system. If a task is not confirmed within its deadline, it is removed from the task vector as to not disturb the progression of the other tasks in the queue. Once finished processing, the task is sent, via cabled connection, to the RSU assigned to send the message back to the vehicle. If that RSU is not directly connected to the Edge, the Migration mechanism handles the correct forwarding. This RSU assignment was already done by the Orchestrator when communicating with the vehicle for the first time and used the route prediction mechanism which will be explained in 4.4.2.

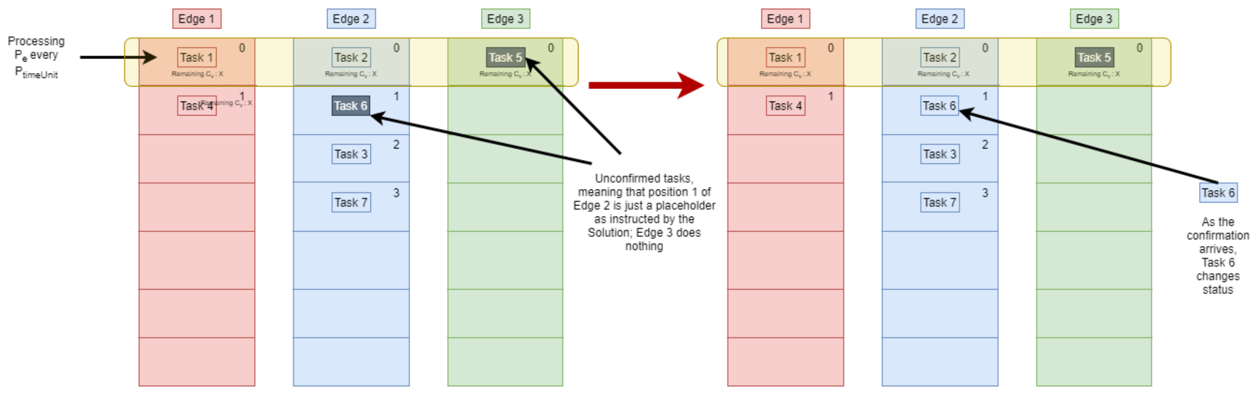

Processing
P_e every
$P_{timeUnit}$
Edge 1
Edge 2
Edge 3
Task 1
Task 2
Task 5
Remaining C_v : X
Task 4
Task 6
Task 3
Task 7
0
1
2
3
Unconfirmed tasks, meaning that position 1 of Edge 2 is just a placeholder as instructed by the Solution; Edge 3 does nothing
Task 6
As the confirmation arrives, Task 6 changes status

Figure 4.5: Task Confirmation in an Edge.

4.2.3 RSUs

An RSU is an entity that resides only in the Data Plane. It is considered a passive entity since it only acts when other entities react to it, meaning that it cannot start any process on its own. In the simulation implementation, each RSU has three Sub-Modules with the following responsibilities:

- **Mobility Sub-Module** - know its geographical location;
- **Application Sub-Module** - receive/send messages from/to Edges; prepare the messages for sending according to message origin;
- **Communications Sub-Module** - Receive tasks from vehicles headed to Edges; complete the retransmission mechanism initiated by the vehicle by sending ACKs.

An RSU's primary function is to be an intermediary between the vehicular users and the resources at the Edge nodes. As such, it has to be able to interpret the incoming message and populate the outgoing message correctly, besides, of course, correctly forwarding it. When an RSU receives a broadcasted message from a vehicle, it checks the assigned RSU ID attached to it, and if it matches its ID, it immediately prepares and broadcasts the ACK response message with the same vehicle ID that was in the original message. This ACK transmission follows the same retransmission protocol first described in 4.2.1, except that instead of sending ACKS continuously, it has a retransmission count, R_{count}, that indicates the program how many times it should send the ACK before stopping, hoping one of those reached the correct vehicle application. After the ACK protocol has been set into motion, the RSU must populate a new Edge message with the required parameters that were forwarded in the original message: vehicle ID, required computation, deadline, and others. Then, the new message is sent upstream through the cabled connection until it reaches the connected Edge. The connection itself has a propagation delay, made possible by the OMNeT++ channel mechanism. The sending of the message has a built-in delay, implemented inside of the code, that models the cabled transmission and routing delays. All of these values are explained in Section 5.2. From then on, the Edge that received the message takes care of its delivery to that message's assigned Edge, as detailed in the next Section.

When the RSU receives the processed data back from the connected Edge, it broadcasts the information to the nearby vehicles with the same modelled transmission delay and with the exact retransmission mechanism used for the ACKs, with a limit in the number of retransmissions.

4.2.3.1 Historical Latency Information for RSU Selection

One important way to reduce the latency a task has is by minimising the propagation delay from vehicle to RSU. This means that if we can always choose, from the RSUs in communication range, the RSU which has the shortest delay, then we can improve the performance of all heuristics that we choose to test. However, the air propagation value is the most volatile of all the values since it depends on distance, obstacles in the propagation path and in-network medium use. The distance is modelled by Veins, and after an internal study of how it works, it was concluded that it does, in fact, model it correctly. In our simulation setup, we do not consider obstacles in the information's path (although Veins has an obstacle shadowing model, this was not used due to the complexity of the tool when trying to use it in our simulations) so that factor was ignored. The most critical point is, then, the medium access problems. If many vehicles request tasks in a very short time window, this is modelled as noise by the wireless receptors in the RSUs and, even though retransmission mechanisms where implemented, this negatively affects the time it takes to send the tasks.

Due to this wide variability of delay that impacts the choosing of the best RSU to send the requesting task, there was a need to develop a new OMNEeT++ network configuration, purposely constructed to gather information on how fast the RSUs, including the same ones used on the main Edge network configuration, could detect a series of tasks from a specific vehicle, among many other requesting vehicles flooding the network. The steps taken were as follows:

1. **RSU Measuring Configuration** - This network configuration is responsible for calculating and storing the delays observed in all RSUs:
    - The fist step is to create a specific Scenario, with no Edges placed and no Orchestrator. We place the RSUs in all of the locations in the Scenario that we will want to collect information from. This will allows us to make a database for those locations, even if not all of them will have RSUs in the real Edge network. We must also use a different Application module, from the one explained in 4.2.3. This one only waits for a specific vehicle's task, and when it is received successfully, i.e, no noise canceled the reception, it saves the information of the time it took for that task to be received, since being first broadcasted in the vehicle;
    - Then we must set up the vehicles' Application modules. There are two types:
        - the first is a specific vehicle, which will be broadcasting the task the RSUs will be listening to. There is only one vehicle of this type. The broadcasts are sent one after the other, until the Simulation stops;
        - the second type is the generic vehicle which only focus on broadcasting messages to simulate network noise, at a random rate between boundaries to make it only

slightly greater than the rate of the first type vehicle. There are many cars of this type circulating, but the exact number determines the "level of traffic" of the Simulation (see Table 5.1). This parameter is used later by the Orchestrator to adapt the decision to different network uses (more vehicles equals more tasks and more medium access issues).

All the vehicles follow random routes and there is no modelling of the transmission times, since it is a common factor between them: we only need the relative differences in propagation time of the RSUs;

- The Simulation is then executed for all three "levels of traffic". Each RSU saves the propagation values to external files. Section 4.4 further details the methods used to store statistics. Several runs for each traffic level are performed so that the values obtained are averages, applicable to more types of routes. Since the distances are already modelled, the values do reflect RSU-vehicle distances averages, and so the propagation values obtained are the only final parameters used when deciding;

2. **Processing Information** - The three files obtained by the runs, each for a different "level of traffic", are then processed in a Python script to obtain the average propagation values by RSU and by traffic.

3. **Edge Simulation** - The Orchestrator in the Edge Simulation network configuration uses this values to reach a good decision of which RSU to send a requesting task:

   - When a task request first reaches the Orchestrator, a routine is executed that updates the Simulation RSUs with the file information. It makes sure that it recognises which RSUs are active and saves in a local database the values for each one;
   - Then, for that task and all subsequent ones, we obtain the current position of the vehicle that requested and:
     – calculate the distance to all active RSUs and determine which are in communication range;
     – among those, find the RSU with the least propagation value, according to the number of vehicles circulating and the historical latency information;
     – save the decided RSU to be sent to the task in case it is accepted by the rest of the decision protocol (if it passes the legality checks).

Figure 4.6 shows a visualisation of some steps taken by the System.

4.2.4 Orchestrator

The Orchestrator is a unique entity that exists solely on the Control Plane. It is the most complex part of the VEC System since it performs critical network management monitoring and hosts several utility functions that provide the necessary support to develop and execute the management heuristics proposed. Its main responsibilities are:

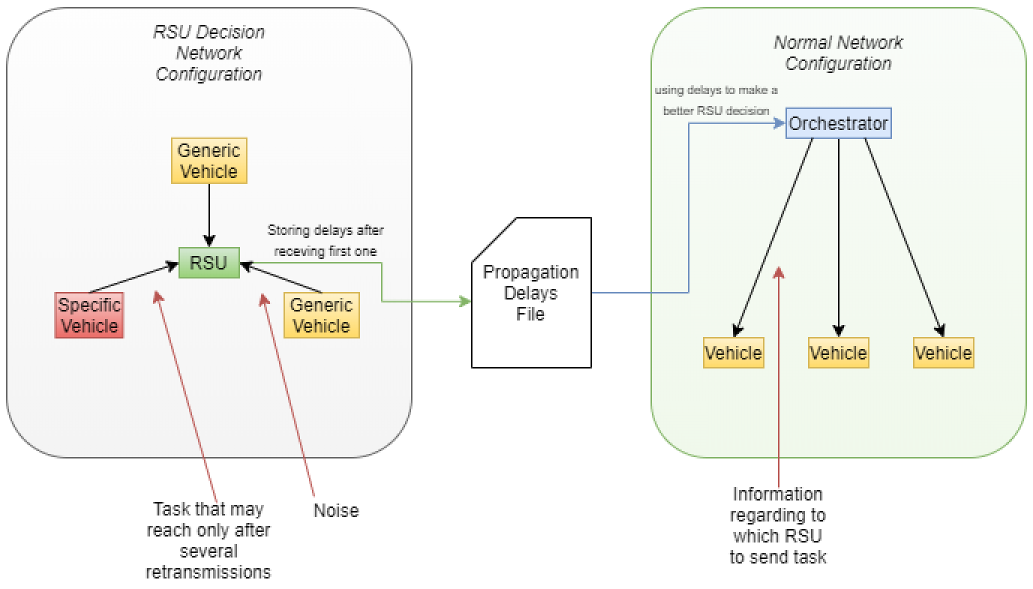

RSU Decision Network Configuration
Generic Vehicle
RSU
Specific Vehicle
Generic Vehicle
Storing delays after receving first one
Propagation Delays File
Normal Network Configuration
using delays to make a better RSU decision
Orchestrator
Vehicle
Vehicle
Vehicle
Task that may reach only after several retransmissions
Noise
Information regarding to which RSU to send task

Figure 4.6: RSU Decision Mechanism.

- Always be up-to-date with available Edge and RSU information;
- Provide Solutions, or task placement decisions, when tasks request service;
- Communicate with the Edges and vehicles, to receive and send information;
- Acquire Simulation Statistical Data.

To help accomplish these goals, we will explain the mechanisms and utility functions that are prepared to work with any System Model and heuristic-based approaches that tackle the provisioning problem in the study. However, the system that records the statistical data is explained in a Section of its own, 4.4.

4.2.4.1 "Solutions" Explained

Although the term has been introduced in Chapter 3, the general idea was vague because this abstract concept can be implemented in vastly different ways. Now, we are going to clarify with a real example of how we implemented a Solution, what it entails and how it is used in the Orchestrator to store decisions.

A Solution is a C++ class uniquely found on the Orchestrator. It is defined as being a vector variable type, native to C++, of another class called Virtual Edge. This Virtual Edge is how the Orchestrator internally stores all the Edges' information. Each Virtual Edge of the vector contains that Edge's unique ID, processing speed, computation capacity and another vector variable, but this time, of a new type called Virtual Task. A Virtual Task is the internal storage representation of the tasks currently in the system, pending or processing. Each Virtual Task contains several variables such as their unique ID, task deadline, the computation required, among other variables that we will later discuss. Virtual Edges and Virtual Tasks are used by the other simulation entities;

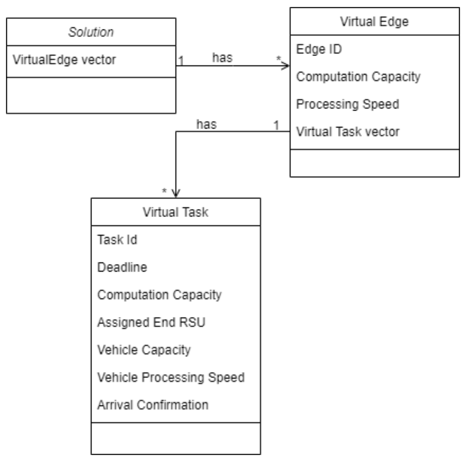


Figure 4.7: Class Diagram of Virtual Edge and Task.

only the Solution class is unique to the Orchestrator. A class diagram relating these three classes can be seen in Figure 4.7.

The Solution class serves a dual purpose, as depicted in Figure 4.8:

- one of the class's objects stores the current network state, i.e., which tasks are in which Edges and their state of completion. We will refer to this main storage as "EdgeInfo";
- copies of the EdgeInfo (other objects of the same class) are created and used by the resource management Heuristics to manipulate the current state of the EdgeInfo until a new Solution is eventually decided.

When a new and better Solution is decided by the management heuristics, it is copied into the EdgeInfo object, and the Orchestrator makes sure its contents are reflected in the respective Edges. This interaction can be seen in Figure 4.9. Conversely, when a new task is requested, the Orchestrator first updates its EdgeInfo object with the Edges' private information before serving the task; this way, it works with the most reliable and up-to-date information.

Note: these information exchanges are all performed in the Control Plane.

4.2.4.2 Solution *Makespan*

When focusing on what a Solution meant in the context of VEC provisioning, in the previous Section, one crucial question arises: how does the System, or in other words the Heuristic approaches explained in 3.5, know that one Solution is better than another? The answer lies in the sum of the latencies of all tasks present in the network, processing or pending.

When a new Solution is generated in one iteration of a heuristic, a method called *calculateMakespan()* loops over all tasks in all Edges and calculates the expected time each task will remain in the Data Plane before being returned to the respective vehicle (local latency). For one task, this time consists of the sum of:

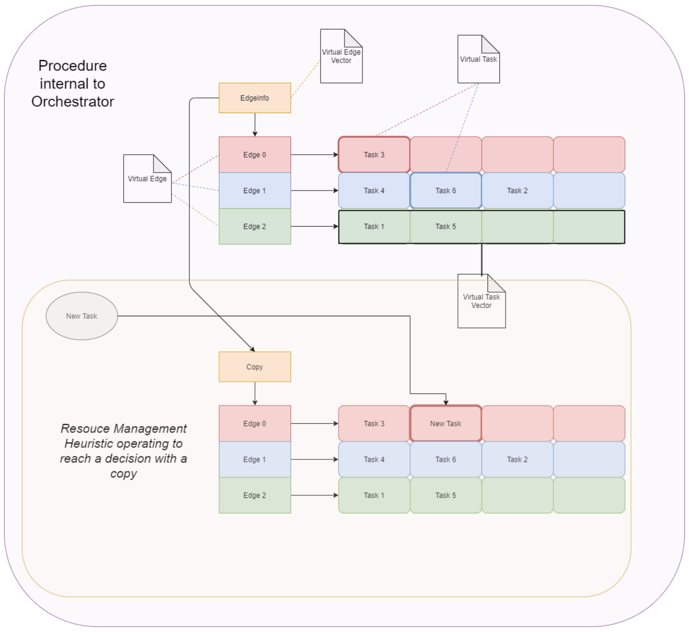


Figure 4.8: Orchestrator Resource Assignment Procedure Overview.

- Its time in the System since the moment it first requested the service;
- The Round Trip Time of the Data Plane communications, detailed in 5.2.1;
- The expected processing time;
- The expected time on the Edge queue, which consists of the sum of the individual delays of the tasks ahead in the queue.

Then we sum all the local latencies to obtain a global latency value called *makespan*. The term *makespan* comes from the operation research field, and it means the time difference between the start and finish of a sequence of tasks. Due to the similarities with distributed systems such as the one in this thesis, we decided to adopt the name as the term used when a quantitative measurement on Solutions is needed to make comparisons and reach decisions. The lower the *makespan*, the better is the Solution since the global latency is what we want to minimise.

4.2.4.3 Legality Checks

Another crucial aspect that arises when obtaining a Solution from an initial one is in what kind of precautions are in place to guarantee that a newly generated Solution is feasible. If no supervision of the results of the heuristics were to be done, this could allow Solutions much better than previous ones, but that could not translate to the reality of the simulation environment since no constraints were taken into consideration. This topic was briefly introduced in Section 3.3. A legal Solution, by definition, has to abide by space and time constraints in the Edge it is assigned: an Edge cannot use more capacity than its maximum capacity value and no task inside an Edge can expect to finish processing over its deadline, this including the Round Trip Time.

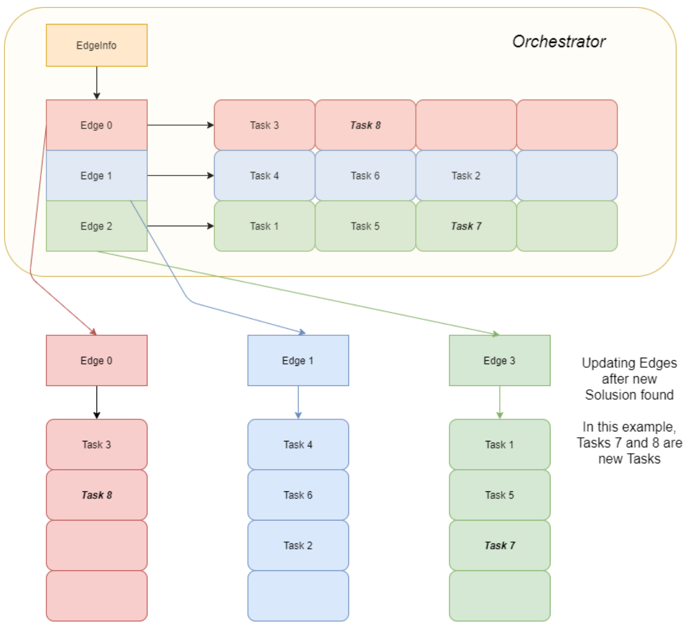


Figure 4.9: Control Plane Solution Exchange Sequence.

The methods that performs this checks is called *checkAllLegal()*. It simply:

- sums the current computation required by each task in an Edge, and if the value surpasses the maximum capacity of that Edge, it deems the Solution illegal;
- computes the expected latency of each task in the task vector using the same principle as explained in the previous Section, and if one is expected to surpass its deadline, it deems the Solution illegal.

Then it repeats for all Edges, and if no constraints are broken, the Solution is legal.

4.2.4.4 The First Solution

In order for a new Solution to be generated, it is a requirement that a base Solution exists since the Heuristics only work on and modify copies of exiting Solutions. Knowing this, we can achieve a quick but efficient initial Solution by ignoring all other existing tasks when deciding the placement of a new task. This is exactly what the *localOptimum()* method does.

When a new task arrives, the Orchestrator has to decide which Edge is the best one for initial placement. It calls the *localOptimum()* which proceeds to iterate over all Edges and tries to place the task at the head of the queue, supposing no other tasks exist. The Edge that can perform the task the quickest is the selected local optimum Edge. If at least one Edge can service the task, then a new method, called *putLegal* takes over, which will be explained shortly. If no Edge can service the task, the Orchestrator assesses the requesting vehicle's capabilities: if the OBU cannot help with the service, then the task is denied service, and it fails to complete.

The *putLegal()* method is called immediately after the *localOptimum()*. It first tries to place the task in the local optimum Edge previously obtained. If that Edge is empty, then it is the best possible outcome for that task, and no heuristic needs to be called to alter the already best Solution possible. If that Edge is not empty, we loop through the Edge and try put the task in the Task vector from the second position onwards (since we cannot move an already processing task) until the first legal Solution is found. If there is no proper placement at that Edge, we try with the remaining Edges until one is found. If by any chance no placement is legal to retrieve a Solution from it, then the task is denied service altogether.

4.3 Implementation of Proposed Management Heuristics

In this Section, we will expand on the concepts introduced in 3.5, explaining how they were implemented using the available tools.

4.3.1 The Baseline Resource Management Heuristic - Simulated Annealing

This approach is easily implemented if the concept of a Solution is well defined, such as the one introduced in Section 3.3. We will now explain the process of how the Orchestrator uses this approach to reach a decision based on newly arrived tasks' demands and the current use of the network.

Firstly, it is important to remember that this Algorithm is only capable of operating after the Orchestrator has already obtained a legal Solution. If no feasible Solution was found during the *localOptimum()* method or if it was found, but no initial Solution was found to be legal, using the *putLegal()* method, then this and all other task placement algorithms will not be called, since it makes no sense working on an empty or non-existent Solution.

The Simulated Annealing approach follows these 5 steps:

1. **Initialisation** - To begin, we must set up a base Solution, as to initialise the working variables: the *current Solution* which is the one passed to this heuristic, the *best Solution* that at the beginning is the same as the current Solution and the *new Solution* which is empty. Other important variables are the *current latency*, the *best latency* and the *new latency*, which will all hold different results of the *calculateMakespan()* method;

2. **Variable Check** - Then, a check on the current Solution must be made to ensure that the first Solution has some task set able to be switched. This means that if there are no tasks at all in the network, or there are but they are the only tasks currently being processed, then there is no point continuing with the execution. Remember that tasks that are actively being processed on edges are unable to be changed to another edge or another position on their internal task vector, by the assumption defined previously on 4.2.2.2;

3. **Normal Operation** - After that, we move on into a while loop that will keep running until a variable called *temperature* is less than an endpoint β, an arbitrary stop value. This is the process that tries to emulate real annealing. Besides this temperature variable, a *cooling rate* determines the speed at which the algorithm executes, or in other words, the amount of time

or iterations remaining until is forcefully stopped. The temperature, which is constantly being reduced from its initial value to β, follows this equation:

$$temperature = temperature * (1 - cooling_{rate}) \tag{4.1}$$

4. **Making a new Solution** - Inside the while loop, the first thing to do is to use the *randomPlacements()* method, which will be explained further ahead on the document, to use the current Solution in order to make a new Solution, which is a slight variation of the first by moving tasks from one place to another. This solution is passed onto *checkAllLegal()* method to ensure it meets all the constraints. If it is not a legal swap, then we continue with the annealing;

5. **Delay Calculation and Analysis** - Finally, we calculate the total delay of the new solution using the *calculateMakespan()* method. If this makespan is better (smaller since we are trying to reach smaller latencies) than the current makespan, we copy the current latency and solution variables with this new one's respective variables. We then compare this new latency to the best latency obtained so far. If it is a better value, we make the same copy operation into the best latency and solution variables as we did before to the current ones. If the new makespan is not better than even the current latency, then there's a probability this worse solution still remains as the current value and solution to the next iterations in order to introduce a diversity that couldn't otherwise be reached. This probability, $P(remaining)$, of changing to a worst solution is greater at higher temperatures and decreases the more the new solution is worse than the current one. It follows the equation:

$$P(remaining) = exp^{\frac{latency_{current} - latency_{new}}{temperature}} \tag{4.2}$$

The *randomPlacement()* method is a simple algorithm that serves to determine which task of a random Edge is placed on another slot of execution flow, which can be of the same Edge or a slot of another. This method is given the solution to alter as an input. As always, a check on this solution needs to be performed in order to detect if the solution has at least one task that can be moved. After, we randomly pick edges until one is found that has a task able to be moved. Internally, we distinguish two types of edge to pick: we call it "type 1" if the Edge has more than 1 task and "type 2" if it has only one but that task hasn't been confirmed to have arrived on the respective Edge yet. This slight difference has an impact when choosing the task to be displaced. From that Edge, we randomly select one task to be moved. Then, we pick without limitations a target destination edge, which will hold the chosen task, as well as the slot, occupied or not, to which the task will go. If the slot is occupied, the task will be inserted between the next slot and the chosen one. To perform the moving operation we use the *taskPlacement()* method: the internal code works operates in the following manner:

- first copy the task to the desired position, using iterators to move into the destination spot;

- After the copy is done, we delete the original task correctly, by virtue of knowing its former location. This is trivial if the destination edge is different from the start edge, but a bit more difficult if the move operation was inside the same Edge, since if one moves an edge to an earlier slot, the original task is pushed down one slot. If one takes into account this slight nuance, then the method is completed, and we return the newly altered Solution.

To better understand this moving operation, Figure 4.10 depicts a normal move operation into an occupied slot.

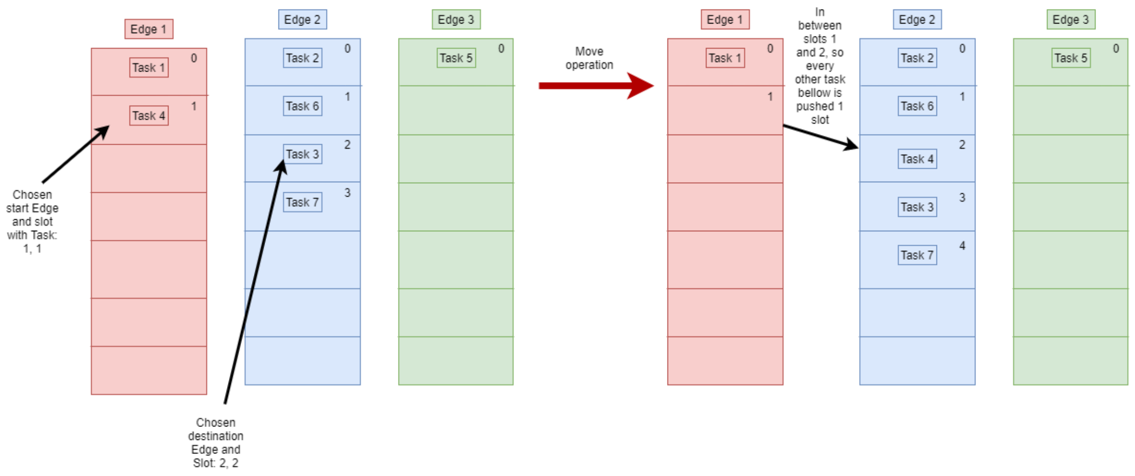

Edge 1
Task 1
Task 4
Edge 2
Task 2
Task 6
Task 3
Task 7
Edge 3
Task 5
Chosen start Edge and slot with Task: 1, 1
Chosen destination Edge and Slot: 2, 2
Move operation
In between slots 1 and 2, so every other task bellow is pushed 1 slot
Edge 1
Task 1
Edge 2
Task 2
Task 6
Task 4
Task 3
Task 7
Edge 3
Task 5

Figure 4.10: randomPlacements() Move Operation.

An algorithmic view of these two methods is demonstrated on Algorithms 3 and 4

Algorithm 3: Simulated Annealing

```
Result: Find new best Solution using task moving until a threshold is reached
currentSolution = initalSolution;
bestSolution = initalSolution;
bestValue = calculateMakespan(bestSolution);
currentValue = bestValue;
if !checkRoutine(currentSolution) then
    return;
end
else
    while temperature > 1 do
        newSolution = randomPlacement(currentSolution);
        if checkAllLegal(newSolution) then
            newValue = calculateMakespan(newSolution);
            if newValue < currentValue then
                currentValue = newValue;
                currentSolution = newSolution;
                if currentValue < bestValue then
                    bestValue = newValue;
                    bestSolution = newSolution;
                end
            end
            else if newValue ≥ currentValue then
                prob = $e^{(\frac{currentValue - newValue}{temperature})}$;
                x = uniform(0.01, 0.99);
                if prob > x then
                    currentValue = newValue;
                    currentSolution = newSolution;
                end
            end
        else
            continue;
        end
        temperature = temperature * (1 - coolingRate);
    end
end
```

Algorithm 4: Random Placement

```
Input: Current task solution: solution
Result: Swap the position of two tasks in random edges and/or their task vector position
newSolution = solution;
if !solution.empty() then
    if !taskChecking() then
        throw cRuntimeError();
    end
    else
        while 1 do
            edgeStart = uniform(0, newSolution.size());
            if newSolution[edgeStart].getTasks().size() > 1 then
                type = 1;
                break;
            end
            else if then
                type = 2;
                break;
            end
        end
        edgeDestination = uniform(0, newSolution.size());
        if type == 1 then
            taskStart = uniform(1, newSolution[edge$_{s}tart].getTasks().size()$) end
            else if type == 2 then
                taskStart = 0;
            end
            if newSolution[edgeDestination].getTasks().empty() then
                taskDestination = 0;
            end
            else if (newSolution[edgeDestination].getTasks().size() >= 1) &&
              (newSolution[edgeDestination].getTasks()[0].isArrivalConfirmation() == true) then
                taskDestination = uniform(1, newSolution[edgeDestination].getTasks().size()+1);
            end
            else if (newSolution[edgeDestination].getTasks().size() >= 1) &&
              (newSolution[edgeDestination].getTasks()[0].isArrivalConfirmation() == false) then
                taskDestination = uniform(0, newSolution[edgeDestination].getTasks().size()+1);
            end
        end
        returnSolution = taskPlacement(newSolution, edgeDestination, newSolution[edgeStart].getTasks()[taskStart],
          taskDestination);
        if edgeStart != edgeDestination then
            auto it = returnSolution[edgeStart].getTasks().begin();
            returnSolution[edgeStart].getTasks().erase(it+taskStart);
        end
        else
            if taskDestination <= taskStart then
                auto it = returnSolution[edgeStart].getTasks().begin();
                returnSolution[edgeStart].getTasks().erase(it+taskStart+1);
            end
            else
                auto it = returnSolution[edgeStart].getTasks().begin();
                returnSolution[edgeStart].getTasks().erase(it+taskStart);
            end
        end
    end
    else
        throw cRuntimeError();
    end
    return returnSolution;
```

4.3.2 The Proposed Heuristic - Rank-Based Approach

We will now explain how the Rank-Based Heuristic was implemented. Besides using once more the concept of Solution to represent the orders sent to the Edges and the requesting task, we need a proper representation of the previously saved network configurations and their assigned past decisions. This is achieved through the use of "Maps".

A Map is a C++ map<string, int> or map<string, double> variable that can relate string variables to specific integer or floating-point values, in order to store information efficiently. If we search by the string, the retrieval of the value is a high-speed operation.

The string variable of the Maps is defined as a concatenation of current Edge utilised capacities percentages, meaning that, for example, if we have three Edges, each with processing and pending tasks that make up 40% of their total Edge capacity, the string would look like:

40;40;40

This will be the way to differentiate between network configurations, which we will call as "Keys". Not all values are allowed in the Keys, however. For convenience, we use ranges of utilization instead of actual values. For example, values in the $[31;40]$ range are represented by '40'. This is done using rounding up operation of the Edge utilised capacity percentage before actually being added to the string. This way, we lose precision but increases efficiency within a confidence interval, since we are simplifying that range of configurations to a single one. This method can be changed to reduce or increase the allowed Keys at will. In the implementation, we use two types of such Map variable: map<string, int>, which we call edgeMap, and map<string, double> named meritMap. The former is used to store the Edge that was chosen to be the best placement location of a task when the string variable represented a specific configuration; the latter is used to store the current Merit, as it was explained in Section 3.5.2, for that same network configuration.

Having these concepts in mind and imagining the arrival of a new task which prompts a new decision request at the Orchestrator, the algorithm, also evidenced in Algorithm 5, is implemented as follows:

1. **Preparation** - The algorithm prepares its initial Solution, which is a copy of the Solution made by the *putLegal()* method;

2. **Key Generation** - The next thing to do is to make the Key of the current Edge configuration, according to the capacity utilisation percentages, as previously exemplified. This Key is immediately saved in the vehicle Application layer for further use;

3. **Key Searching** - Then we need to see if the values of the capacities are known because if they are, we skip the "learning phase" and take advantage of the algorithm, as designed;

4. **Key Utilisation Choice** - We search the edgeMap by the Key, and if we find a match, an exact or approximate representation of the Edges' state, the next step is to generate a random value x which will be used to compare with the value of the Key in the meritMap. If the random value is lower or equal than the Merit, we indeed create a new Solution by placing the task at the end of the queue of the Edge that was deemed the best selection to that Key, stored in the edgeMap. Best choice means that it achieved the lowest global latency in the Learning Phase. If x is bigger, we have to recalculate the best Edge to this Key once more. This learning operation, as implemented, tries to put the requesting task in as many placements of the queues in the Edges as possible, without letting it be a significant past of the total delay; then it calculates the resulting makespan (if it is a legal Solution) for each Edge. This way of task placement was the one considered due to the fact that all tasks in the simulation are assumed to be the same, since there is only one application, 5.2.1.1, with particular requirements being simulated. The Edge that got the best makespan is assigned as the new value of the Key in the edgeMap. If no Solution is found, the algorithm returns without changing the initial Solution;

5. **Learning Phase** - On the other side of the spectrum, if a Key is not found when first searching the edgeMap for the current configuration, the learning phase described is immediately applied, and the value is set to that new Key, along with the setting of initial value to the respective meritMap position (more details are given below). If no Solution is found, then the Key is not inserted in the map variable and the algorithm returns without changing the initial Solution;

6. **Conclusion** - If a Solution is found, either by using a previous result or a new one, the initial Solution is replaced, and the algorithm returns successfully.

The second part of this process is the analysis of the results and Merit adjustment of the configurations. There are two moments when the Merit of a Key is modified:

- When first assigning a base Merit value to the Key, when it is created. This base Merit must be between 0 and 1, 1 corresponding to always use this first assigned solution no matter what and 0 to only use this solution once, which is now. The value in between these two must be carefully chosen if we want a stronger or lesser first "impression" of the network to be carried onwards. A bigger value makes it harder for future recalculations and variability in the simulation's initial stages, while a lesser value makes it too volatile.

- When a processed task arrives back to the vehicle user. The vehicle signals the Orchestrator that it can check what Key it was used to decide the task placement (stored in the vehicle in the second step) and the Orchestrator proceeds to adjust the Merit of that Key based on the amount of time it took to complete and the average latency registered so far across all finished tasks. There are two important notes in this step: the first is that we only start taking into account the average latency after a certain amount of tasks, n_{setup}, have finished since making decisions and changing values with an average calculated based on a small sample size lessens the mathematical significance of the whole process; the second note is that the increases and decreases in Merit, if a task performed above or below average respectively, are done in a proportional and bound way, meaning that the more a task performs better/worse than the average, the more the Merit increase/decrease, as a way to benefit/punish Keys based on results, but always only up to a maximum and minimum Merit limits, *upperLimit* and *lowerLimit*, imposed by the design considerations. Figure 4.11 exemplifies the flow of the Merit Calculation process.

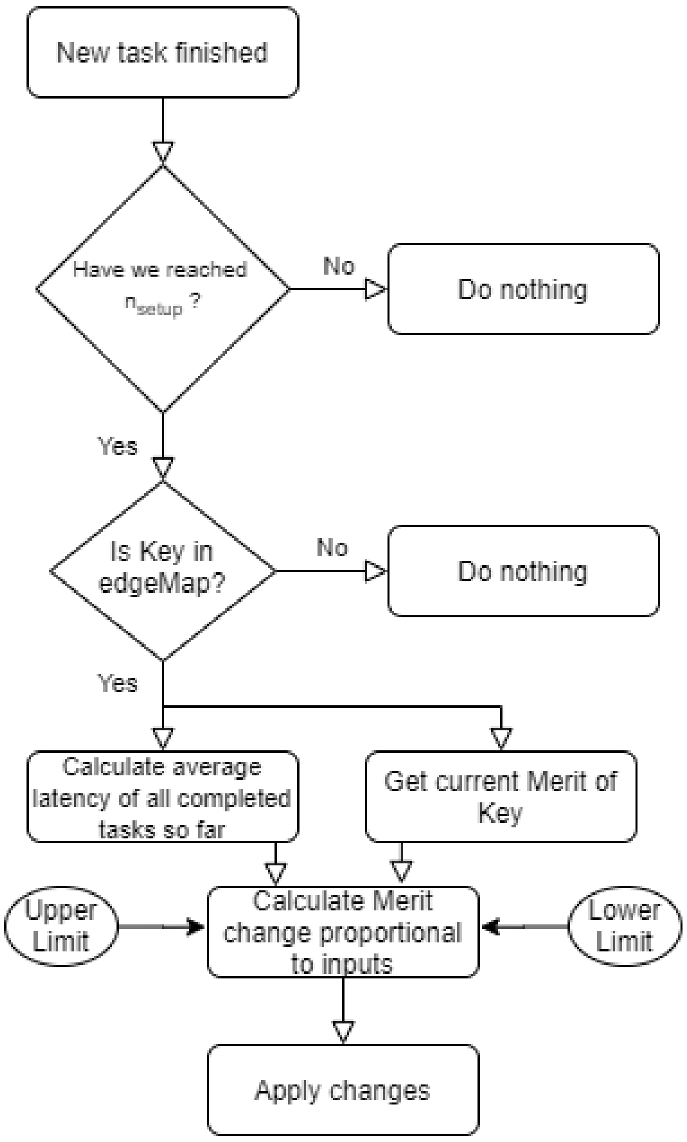

New task finished
Have we reached n_{setup} ?
No
Do nothing
Yes
Is Key in edgeMap?
No
Do nothing
Yes
Calculate average latency of all completed tasks so far
Get current Merit of Key
Upper Limit
Calculate Merit change proportional to inputs
Lower Limit
Apply changes

Figure 4.11: Merit Calculation process

Algorithm 5: Rank-Based Approach

```
Input: Current task solution: Solution; requesting task; vehicleApp
Result: Obtain a Solution based on previous network configurations/ edge resource utilisation
error = 100000, bestEdge = -1, bestValue = error, newValue = 0, capacitySum = 0, capacityPercentage = 0;
key = null, newSolution = null, bestSolution = Solution;
for $i \leftarrow 0$ to $Solution.size()$ by 1 do
    for $j \leftarrow 0$ to $Solution[i].getTasks()$ by 1 do
        capacitySum += Solution[i].getTasks()[j].getComputationRequired();
    end
    capacityPercentage = capacitySum/Solution[i].getComputationCapacity();
    rounded = std::round(capacityPercentage*100);
    key+= std::to_string(rounded) + ";";
    capacitySum = 0, capacityPercentage = 0;
end
vehicleApp->setKey(key);
if edgeMap.find(key) == edgeMap.end() then
    Learning Phase;
    for $i \leftarrow 0$ to $Solution.size()$ by 1 do
        newSolution = taskPlacement(Solution, i, task, Solution[i].getTasks().size());
        if checkAllLegal(newSolution) then
            newValue = calculateMakespan(newSolution);
            if newValue < bestValue then
                bestValue = newValue;
                bestSolution = newSolution;
                bestEdge = i;
            end
        end
    end
    if bestValue == error then
        return -1;
    end
    End Learning Phase;
    edgeMap.insert(std::pair<std::string, int>(key, bestEdge));
    baseMerit = par("baseMerit");
    edgeMap.insert(std::pair<std::string, double>(key, baseMerit));
end
else
    if edgeMap[key] == -1 then
        return -1;
    end
    else
        x = uniform(0.01, 0.99);
        if x <= edgeMap[key] then
            bestSolution = taskPlacement(Solution, edgeMap[key], task, Solution[edgeMap[key]].getTasks().size());
            if !checkAllLegal(bestSolution) then
                return -1;
            end
        end
        else
            Same as Learning Phase;
            edgeMap.insert(std::pair<std::string, int>(key, bestEdge));
        end
    end
end
return bestSolution;
```

4.4 Complementary Aspects of the Simulation Framework

4.4.1 Acquiring Statistics

In order to analyse and display the quantitatively results of the Performance Metrics considered in 6.1, which shows us the effectiveness of the heuristics and the whole Edge node management system, we had to prepare the Simulation with ways to collect the large amount of information we wanted, as well as transfer it to a tool which could process such amounts in a very fast way:

1. **OMNeT++** - Firstly, when we want to gather some information, the best way to do it while the Simulation is running is by using a variation of the Signal mechanism introduced in 4.1.3. The overhead it imposes on the executing operations is minimal and is well adapted to constantly be collecting information. When we have a new value to add to statistical analysis, we simply emit a signal with that value from the module where is was created. Then a receptor of that signal, located in the main module of the program (Network Configuration Module), adds this value to a statistical storage. OMNeT++ already gives us the ability to store the values as vectors (all the values obtained over time) or scalars (single values). Furthermore, we can specify mathematical operations to perform before adding to the statistical storage, such as simple means or more complex custom operations. All of the values are stored in a specific type of file which allows us to see not only the values but some simple plots over time or even histograms. The limitations are many, however, and so we opted to export the information as a *.csv* to be used by other, more powerful processing tools;

2. **Python** - This programming language (81), due to its simplicity but vast and powerful information manipulation options, was the one chosen to perform the data aggregation and graphical representations we needed. In a script, we can import the *.csv* file previously generated and save it in a variable. Then we pinpoint the information we want in that variable and we start operating over the data:

   - if the information gathered is relative to multiple runs using the same Simulation Parameters (as a way to get statistical significance), we can make arrays or lists that correctly store the values and combine them into meaningful averages;
   - export to another *.csv* single values such as averages, rates, or absolute values, needed to have a quick representation of global variables;
   - make boxplots, histograms or over-time graphs that allows to perceive important details of a set of data and compare at a glance the results from multiple runs or different sets of Simulation parameters.

4.4.2 Mobility Trace Generation Procedure

One of the parts described in 3.1 referred the possibility of the vehicle's routes to be known *a priori*, meaning that when vehicles appear in the Simulation, the system knows the path they will take and the RSUs that will be reachable throughout it. This knowledge allows for an informed decision of which RSU the processed data must go through so it can reach the vehicle the fastest way possible by reducing the information propagation delay.

Decisions are only taken when the the Orchestrator receives a new task. Thus, the routes' information needed must already be known beforehand. This information is gathered and prepared in the following way:

- Before running the actual Edge Simulation properly, with the routes created by SUMO, a smaller OMNeT++ network configuration, specifically made to accommodate this mechanism, must be run with those same routes:
  - The vehicles follow the SUMO generated routes and each second, they save their current positions to their private map variable (82), which can save data in pairs. These pairs correspond to the moment in time when the save action was done and the X and Y coordinates of that moment. This continues to happen until the vehicles stops their simulations;
  - Then, before exiting the program, the vehicles signal a Trace Manager class. This particular entity simply writes to a file the map of the vehicle that finished the simulation along with the vehicle ID. It repeats for every vehicle until the whole simulation is completed;
  - The end results is a traces file with all vehicles' coordinates over their simulations period.
- After obtaining the paths' traces, we can run the actual simulation of the VEC system:
  - The first time the Orchestrator receives a task, it executes a method that converts the information in the files into a map variable. By nesting another map inside the first map, we can effectively store and retrieve the coordinates of a vehicle in any recorded moment in time;
  - After that, and to every new task received, just before sending the decision to a vehicle, we retrieve the coordinates of the vehicle at the expected instant of time that the information processing is complete and considering the deadline of the task. If an exact moment is not found, we consider the closest recorded moment, rounding up;
  - The position allows the Orchestrator to know which RSUs will be in communication range, and depending on the proximity to the vehicle and the estimated latency of the individual RSUs propagation delays, using the same mechanism introduced in 4.2.3.1, the best RSU is chosen to be the one delivering the processed data.

In the end, this mechanism is a simplification of more complex pattern recognition systems. Despite having to run two simulations instead of just one, with every new set of random routes, the information is important in achieving smaller latencies. Figure 4.12 puts into perspective the process.

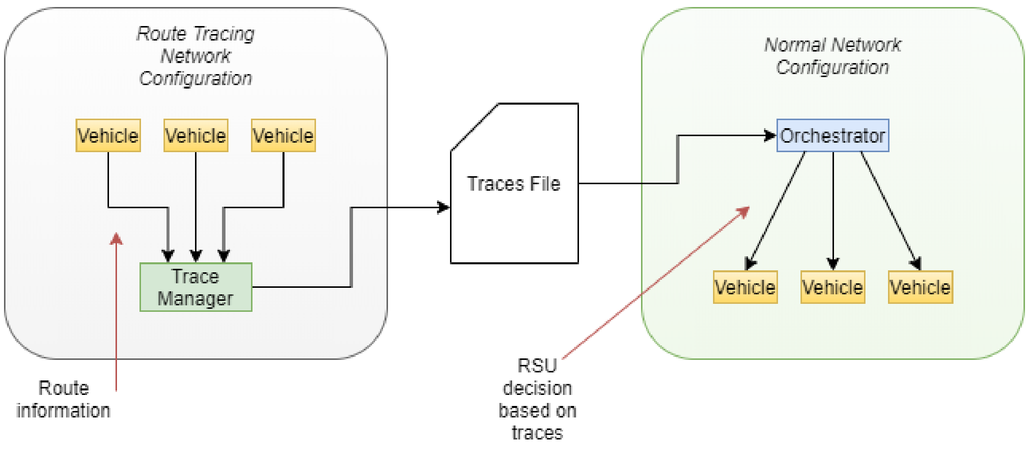


Figure 4.12: Trace Mechanism.

4.5 Final Remarks

In this Chapter we described the Simulation system, how all of the system entities were implemented and we detailed the Heuristics developed.

The main takeaways of this Chapter are:

- Data Plane communication between entities is the only type of communication being actively simulated;
- the Orchestrator first decides and then distributes the Solution to all Edge Nodes, so that each one knows what and when to execute;
- vehicles can request multiple tasks pertaining to the same application, with 1 second interval between each request;
- a Retransmission Mechanism was implemented in the RSUs and vehicles to prevent data loss;
- a task being processed cannot be moved to another Edge in the decision process of the Orchestrator. This prevents data corruption;
- a Solution only exists in the Control Plane;
- historical latency information for each RSU allows for a better decision of which RSU to choose from when sending task to network, lowering the global latency.

Chapter 5

Scenario and Simulation Parameters

This Chapter has the purpose of explaining the specific Simulation Framework, the additional assumptions considered and detailing the origin and argumentation behind all the values used in the Simulation.

5.1 Simulation Framework

We begin this Section by grabbing the reader's attention to Figure 5.1. It depicts the System Model fully realised, at the exact moment a vehicle broadcasts its task to all RSUs in range. A quick comparison with Figure 3.1 illustrates the similarities between the two and the effort made to bring to reality the concepts previously introduced.

We revisit the Scenario description discussed in Section 3.1, and we will now present the specificities and assumptions of this System Model:

- We abstract the Control Plane communication, meaning that when the Orchestrator communicates with the Vehicles and the Edge nodes (and vice-versa) there is no simulation of the communications between them;

- It is important to know that the road topology greatly affects the results of the case study, because the frequency of intersections or the road length can greatly impact the minimum or maximum velocity a car can reach, which in turn affects the maximum delay a task can have. The chosen road topology is the **Manhattan Grid model** (64), as several works mentioned in 2.3 also used. This model is simple, yet allows for a wide control of all the other parts involved and it is easy to grasp visually. Each "square" of the grid is 100m of side: the actual Manhattan Island (where the topology name gets its name from) is 21.6 km long and 3.7 km wide at its widest, but only about 10km long is the grid-like streets of what we expect. So, a grid of 9x9 of these squares is a realistic vehicular scenario, making it 900 meters long and wide. See Figure 5.2 for a street like view of what it could be like in a real scenario;

- All edges of the roads are considered to be **Entry/Exit Points**. These act as the beginning and end of a vehicle's path. A vehicle is only allowed to spawn from one of these areas, and

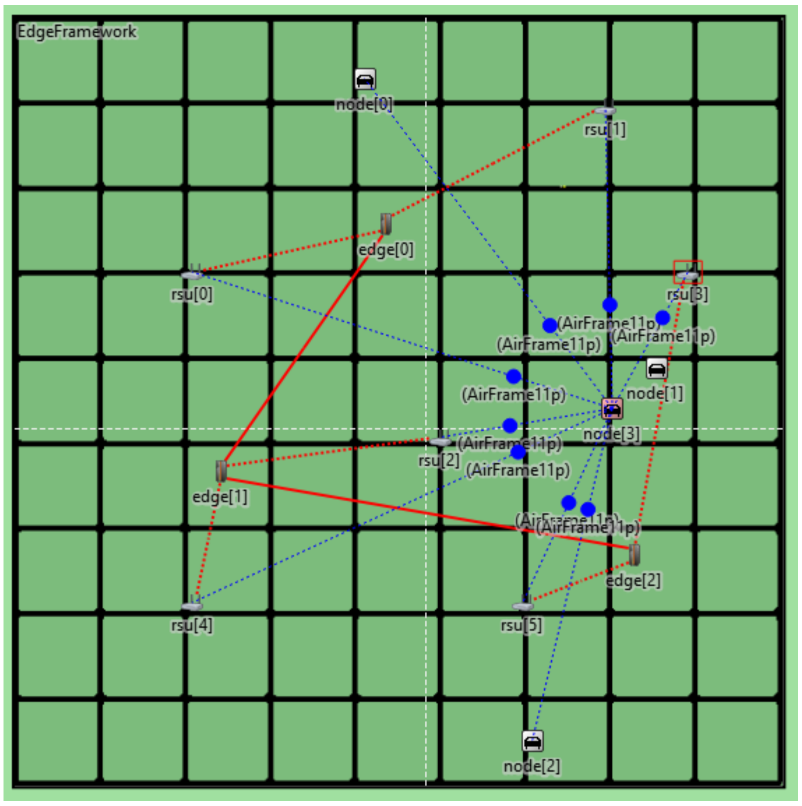


Figure 5.1: System Model Realised.

it must make its way to another. The location of these points at the beginning of a simulation is randomised as to simulate a natural city section of a city. The arrival rate of the vehicles at the edges of the map follows a **Binomial Process which is approximated to a Poisson Arrival Process**, meaning that the arrivals rates of the vehicles, over some period of time (e.g., seconds), are randomly sampled from a Poisson distribution, which is realistic in this scenario. This rate of arrival and the total amount of vehicles that spawn during a simulation are vital factors that influence the results obtained;

- When a vehicle is spawned, a **pre-defined route** between Entry/Exit points is assigned to it, meaning that the routes are known *a priori* and the planning of the best RSU to send information can be made. This planning was explained in 4.4.2;

- The maximum speed of the vehicles, a customisable parameter specified in the Values Section (5.2), does not mean that the speed is constant. We will use the default SUMO car-following model, the Krauss-model (83), meaning that each vehicle must be able to stop safely even if its lead vehicle suddenly brakes to a full stop;

- A vehicle application can send N_{tasks} which represent the tasks needed to meet the application requirements. If more than F_{limit} messages do not meet the deadline after returning from being processed, that **whole vehicular application fails** and it counts as a failed/refused operation for statistical purposes;

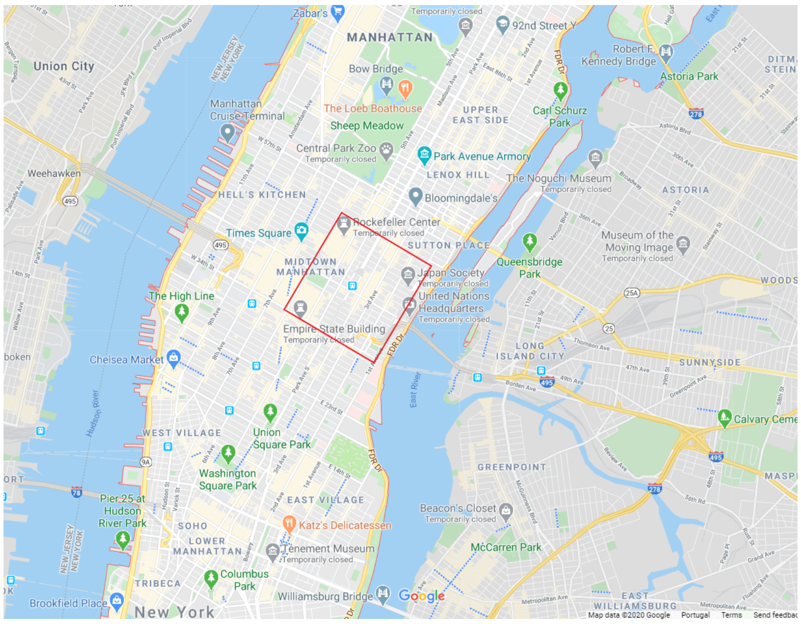


Figure 5.2: Street like view of Manhattan taken from Google Maps.

- The maximum deadline of all tasks requested by the vehicles is **1 second**, due to the nature of the application considered;
- The vehicles communicate with the RSUs using the IEEE standard for Wireless Access in Vehicular Environments, **802.11p** (84);
- Each cabled connection (red line in Figure 5.1) is considered to have **one switch** in the middle that imposes routing processing delay. We represent the switch as its delay and not as a physical component;
- Given the reliability of the cabled network infrastructure, we will assume that there is **no packet losses** in this medium;
- Each Edge can connect with **up to two other Edges**, using cabled backhaul;
- All Edges are considered to have **the same GPU**;
- One Edge can have up to **two connected RSUs** and each RSU is connected to **only one Edge** at a time, using cabled backhaul;
- In the cabled networking infrastructure of our scenario, the length of Edge-Edge and Edge-RSU links are very short and the difference in the associated propagation delay is negligible. Hence, for simplification purposes, we assume **a single distance for all wired links**. This distance is fixed and it is explained in Section 5.2;
- In total, the final system model for statistical analysis consists of **3 Edge nodes and 6 RSUs.** The Orchestrator has to manage their resources to provide service to meet the demands of vehicles. The exact locations can be seen in the Figure 5.1;

5.2 Simulation Values Utilised

It is important to use realistic values when dealing with real-time systems. That simple fact alone applies heavy weight on the relevance of the results that can be deduced and adds importance to the conclusions' applicability in a real scenario. When searching for these values, one has to delve as deep as it can to find trustworthy information but with the presence of mind that there will always be restrictions on that information and that achieving a 100% realistic scenario is more complex than it can be managed.

This Section will present an explanation of all of the values utilised throughout the work. They are parameters of the developed framework and thanks to an existent configuration file in OMNeT++, they are located in a single location, ready to be modified at the user's will. These values are based on literature research, on physical equipment and on applications' delay estimations.

5.2.1 Values of the Communication and Computing Time Components

This thought applies to the Communication Values now presented. These values refer to the values used when trying to find the Round Trip Time (R_{tt}) of a task request.

This time can be defined as the sum of the Outgoing Time (T_o), the Application Processing Time (T_{appP}) and the Return Time (T_r):

$$R_{tt} = T_o + T_{appP} + T_r \tag{5.1}$$

where

$$T_o = T_{cp} + T_{ct} + \theta(T_{ap} + T_{at} + T_{ma}) + T_{rp} \tag{5.2}$$

and

$$T_o = T_r \tag{5.3}$$

T_o is the sum of the Cabled Propagation (T_{cp}) and Transmission delays (T_{ct}); the Air Propagation (T_{ap}) and Transmission delays (T_{at}), as well as the Medium Access Delay (T_{ma}), which are all multiplied by the probability of packet retransmission (θ); and the Routing Processing Delay (T_{rp}). A transmission delay is how long it takes to get all the bits of information into the cable. The propagation delay is the time it takes one bit to propagate from one end to the next in the connection. A routing delay is the amount of time taken by the switch to process the packet header and to know where it must forward the information. T_o is considered to be the same as the T_r in Equation 5.3; despite having stochastic components (i.e., variability), over long periods they tend to be the same in average. This allows for this approximation of the two components.

When talking about Round Trip Times, one normally assumes the go-and-back trip of a new block information (in this case, a frame), and not the aggregated value of multiple frames. For convenience, in the following discussion, we refer to the sum of RTTs of all packets due and transmitted over the course of a second. E.g., if 10 frames/packets are due a given second, RTT and all delay metrics (unless specified otherwise) refer to the sum of RTTs for those 10 frames/packets.

5.2.1.1 Application Data Size

This value does not appear explicitly in the above equations, but it is used to calculate many of them since various delays require to know the amount of information a vehicular application will need to send to be processed at the Edge nodes. It is important to note that it is assumed that the amount of information used when sending to the Edge node is approximately the same as the amount of information returning to the vehicular user, after processing. This assumption is reasonable since processed information is typically less than the raw data, and as such, we are simulating a less favourable scenario where we must move almost the same information back and forth since the application so demands.

The real-world application used throughout this work is a Real-Time Object Detection system, implemented with the algorithm YOLO (You Only Look Once) (72). This application requires Image Processing capabilities. Thanks to the information about the time taken to process one frame, the resolution of the images that were tested, and the number of FLOPS it needs per frame, we can compute all other values needed to know the timings of the whole process, from the moment this application is requested in a vehicle, to the arrival of the processed images back to be used by YOLO.

We assume that this application takes 24 frames (images) each second (24 FPS) and needs to have that data processed in order to detect objects in the vicinity of the user. 24 FPS is the value used since it is the minimum speed needed to capture video while still maintaining realistic motion. Since it refreshes the capture every one second, **the maximum deadline of all tasks of our system, T_d, is also 1 second**, lest all frames lose temporal significance. Each one of the frames is an image with a resolution of 320*240 (Quarter Video Graphics Array), 3 pixels of colouring, which corresponds to the YOLOv3-320 Model (a more detailed data set which this model operated on can be found in (72)).

The application data size is, for 24 frames:

$$320 * 204 * 3 * 24 = 0.659MB \tag{5.4}$$

Note: To convert from pixels to MB, just divide the pixels by (8*1024*1024).

If we consider that the application uses the H.264 video compression technology (85), or codec, the total amount of data to transmit diminishes by up to 50%, a value we are going to use.

$$\frac{0.659MB}{2} = 0.33MB \tag{5.5}$$

One important aspect to take into consideration is that each packet that is sent from the vehicle to the RSUs needs to have a TCP/IP Header which increases the total information sent. Each header is 20 bytes minimum (86). Another aspect is that Ethernet links restrict the max transmission unit (MTU) to 1500B. Using these values, the total Application Size for a 1-second capture is

$$0.33MB + 20B * \frac{0.33MB}{1500B} = \mathbf{0.3344MB} \tag{5.6}$$

5.2.1.2 Cabled Propagation Delay

There are two types of cabled connection in the system: between RSUs and Edge nodes and between Edge Nodes. Both types are considered to be the same in all aspects except when used in the internal logic of the program. For our concerns right now, the type of cable is the same: fibre optics. Using fibre optics, information takes 5us to 5.5us to travel 1 km, since it travels between 180,000 to 200,000 km/s. Knowing that each side of the system model square scenario is 900m and that the worst case would be travelling at the slowest pace in a diagonal connection, the value used will be approximated to **5.5us**.

5.2.1.3 Cabled Transmission Delay

According to (87), the Cabled Transmission Rate of a Fiber Optic Cable, Multi-Mode, up to 1Km maximum distance is 1Gbps(125MBps). Since the connections used between Edges and RSUs will have about 300 to 600 meters, this value is adequate. With this information, and knowing the Application Data Size, we can calculate the maximum delay:

$$T_{ct} = \frac{0.3344MB}{125MB/s} = \mathbf{2.68ms} \tag{5.7}$$

5.2.1.4 Air Propagation Medium Access Delays

In Equation 5.2, we separated the components of the typical Air Delay. Two of those components are the Air Transmission Delay, which is a well known physical propriety, which depends on the medium the information is travelling; the other is the Medium Access Delay, that depends on the number of requesting entities at any given moment in the network.

The former value is the only one that Veins models, and so, we did not have to manually add it in our Framework, in contrast to all the others described in this Section. The latter depends on the probability of retransmission, θ and its exact value depends on the road and network conditions. However, knowing an approximation of this value in theory is valuable, because it helps us reach the theoretical value of the Round Trip Time, useful in some of the Orchestrator calculations.

The method used to find the average of the **sum of these two components** is by averaging the values obtained by the special network configuration explained in 4.2.3.1. Here we will only present the average value obtained after several iterations of the Simulation, across all Traffic Levels: **240us**.

5.2.1.5 Air Transmission Delay

For this delay, we use the maximum transmission rate of the IEEE 802.11p standard, 27Mbps. (88). We divide the application size by this value.

$$T_{at} = \frac{0.3344MB}{27Mbps} = \mathbf{99.1ms} \tag{5.8}$$

We had to incorporate this value into the Simulation entities that use it.

5.2.1.6 Routing Processing Delay

We take a representative switch, namely a C9200-24T to inform us about the routing processing delay (89). It is a switch with no stacking and good bandwidth specifications between the ones commercialised by Cisco. The forwarding rate of the chosen switch, measured with 64 byte packets, is 95.23 Mpps(million packets per seconds).

$$\frac{0.3344MB}{64Bpacket} = 5225packets \tag{5.9}$$

$$Trp = \frac{5225packets}{95.23Mpps} = \mathbf{55us} \tag{5.10}$$

This value is very small compared to the others.

5.2.1.7 Application Processing Time

This time depends on the type of GPU used to process the information, since a more powerful one can shorten the processing time substantially, which is the biggest component, by far, of the total latency when sending a task to an edge. According to the YOLO website, one frame requires 140*10e9 FLOPS; the GPU used to make the measurements was an NVIDIA Titan X Pascal (90), which as a Theoretical Performance of 10.97*10e12 FLOPS.

Note: Floating-Point Operations per Second (FLOPS) is a standard measure of computer performance, useful in fields of scientific computations that require floating-point calculations (71). It is a more accurate way of measuring than the instructions the per second variant.

According to these 2 values, the theoretical processing time is approximately

$$\frac{140 * 10e9FLOPS}{10.97 * 10e12FLOPS} = 13ms \tag{5.11}$$

which is less than the 22ms per frame reported by YOLO's measurements, although being in the same order of magnitude. This difference can be explained due to memory accesses delays that are not counted when making a theoretical approach only. The ratio between the values is

$22/13 = 1.6923$ that needs to be applied in the implementation work. Using the measured values, T_{ap} is:

$$TappP = 22ms * 24FPS = \mathbf{528ms} \tag{5.12}$$

5.2.1.8 Final Round Time Trip value

Revisiting equations 5.1 to 5.3 with the values of the previous subsections, over 1 second:

$$T_o = T_r = 5.5us + 2.68ms + 240us + 99.1ms + 55us = 102.0805ms \tag{5.13}$$

$$R_{tt} = 102.0805ms + 528ms + 102.0805ms = \mathbf{732.161ms} \tag{5.14}$$

It is a good expected value since 1 second is the maximum delay and in reality, other smaller delays, such as the previously mentioned theoretical-only concept of a central Edge node which is not modelled, will increase this number by a few more milliseconds.

5.2.1.9 Other Design Considerations

A first study with only 10FPS and with no colour (1px - grey-scale) was made, and the result for the R_{tt} was of 234.67ms. After that, the frames were increased to 24FPS with no change to the colour, and the resulting R_{tt} rose to 563.2ms, which is almost the same ratio of the FPS rise in between tests. Then it was incorporated 3 colours instead of grey-scale, and the result is the one previously thoroughly demonstrated before. It is an important realisation that although we multiply the number of pixels by 3, it only gives a 1.3 times increase on the final result. This means that we could achieve slightly more FPS if needed if we do not need the images to be coloured.

Another study was done to the resolution used. The YOLO application website gives two more measurements, for 416*416 and 608*608 aspect ratios of the images. We can increase the resolution at the cost of the number of frames. For example, if the latter aspect ratio is used, it incurs a 51ms per-frame processing time, taken directly from the source, which would quickly go over the 1s deadline. It is clear that the FPS need to be reduced substantially because of this and since data sizes are 4.8 times larger than the 320*240. This would greatly impact most of the values calculated above. An easy conclusion can be made with this alone: the FPS would be too low to any real application to take good advantage. To conclude this part, below are the calculations performed to the 608*608 resolution images and X FPS, using the same methods as above, but now using this X variable instead of 24 FPS. Imposing that the value of the total delay must be less than 800ms, to be close to the previous result (with air and cabled propagation delays remaining the same):

- Application Size - 0.066982MB*X

- Cabled Transmission Delay - 0.535856ms*X
- Air Transmission Delay - 19.8ms*X
- Routing Processing Delay - 11us*X
- Application Processing Delay - 51ms*X

$$T_o = T_r = 5.5us + 0.535856ms * X + 240us + 19.8ms * X + 11us * X = 20.6ms * X \tag{5.15}$$

$$R_{tt} = T_o + 51ms * X + T_r =< 800ms \tag{5.16}$$

Realising the above equation results in $X < 8.67FPS$, which means at most we could have 8 FPS, which is too low of a value, as expected.

5.2.1.10 Processing Speed of OBU

To complement the discussion of the Application Processing Time performed in 5.2.1.7, we will now introduce the concept of OBU Processing Speed.

Every vehicle in the simulation is a Smart Vehicle, meaning that they are equipped with an On-Board Unit (91) capable of processing vehicular applications, if need be. Despite the prevalent usage of the Edge Servers in the heuristics developed, due to having greater processing capabilities, the system may use the resources of the OBUs if either of the two conditions explained in 4.2.1 are met.

Because of this, it is important to clarify how many FLOPS a typical OBU (92) possesses, to determine the time it takes to complete a task and to verify if the chosen Image Processing Application can be run in the vehicles at all. After a research phase, it was concluded that the processing power of a typical OBU is not considerably larger than an available Raspberry Pi 4 Model B(4GB) (93). This will be our best-case scenario since this device is quite powerful for its size. According to (94), this machine can deliver 13.5 GFLOPS. Using Equation 5.11 and maintaining the same Application FLOPS:

$$\frac{140 * 10e9FLOPS}{13.5 * 10e9FLOPS} = 10.37s \tag{5.17}$$

This value is considerably larger than the 1-second deadline already established, and as such, **for this application in particular, no task can be done on the vehicles**. Even so, the developed edge management framework is prepared to handle smaller applications or parts of applications that may use the OBU if it meets their specific deadlines.

5.2.2 Working Values

This Section holds all other parameters considered throughout the document. It is advised to read this Section alongside their relevant discussion in the document, as to fully grasp the concepts introduced and their practical use in this work. The relevant Section will be indicated when appropriate, to ease the search.

5.2.2.1 Simulation Values

The following five values were introduced in Section 5.1.

- Number of Edges - 3
- Number of RSUs - 6
- Maximum number of RSUs - This value depends on the number of intersections. For the Simulation environment created, the maximum is 16.
- Number of Switches per link - 1
- Max "Transmission Range" - As explained in 5.2.2.3, the value used is 430 meters.

Regarding the values intrinsic to the OMNeT++ simulation environment and the Veins framework that runs over it, we can find those which control the simulation configuration itself and those which modify the C++ applications that emulate layers of the OSI Model. The full list of parameters is too extensive to analyse here, so, only the most important ones that are not relevant to other sections will be described next:

- **sim-time-limit** - it defines the time the simulation a real-world scenario. The value is set to 300s, which is more than the 100 to 150 seconds the vehicles need to finish sending their tasks and to retrieve the results;
- **SumoManager.updateInterval** - it corresponds to the time step the simulation uses to acquire new position and speed values from the SUMO application. The time step is 1 second, meaning that only by second the vehicles can move in the simulation;
- **nic.mac1609_4.txPower** - it is the value used by the Media Access Control Layer to know from how far signals must be recognised or discarded when receiving wireless packets. It is greatly important to the operation of the assignment of the tasks to RSUs. Its value is 20mW, as it will be explained shortly;
- **playgroundSizeX and playgroundSizeY** - they define the size of the Manhattan grid. They are both 1000 meters, making the grid a square;

- **SumoManager.launchConfig** - this parameter holds the XML description of the .net.xml and .rou.xml created by SUMO to be used by VEINS TracyMobility Model. The .net file describes the configuration of the roads and the .rou file describes the routes of the simulated vehicles;
- **nic.phy80211p.usePropagationDelay** - this is a boolean value, which we set to true, to allow Veins to simulate Propagation Delay in the model.

5.2.2.2 Propagation Modelling and Parameters

In our simulation scenario, we do not model complex propagation phenomena caused by obstacles, e.g. attenuation or multipath from buildings, as would be expected in urban and suburban scenarios. However, using the typical transmission power for a IEEE 802.11p setup of 20 to 23dBm (100mW to 200mW), a transceiver positioned anywhere in the world experiences lossless communication to any other terminal. Hence, to compensate for the inexistence of attenuation by the buildings, we **lowered transmission power to 13 dBm (20mW)**. This can be seen as a scale factor; the alternative would be to scale up the simulation world to be of several kilometers wide, which we deemed unnecessary;

5.2.2.3 Propagation Model Used by the Orchestrator

The algorithm assigns RSUs to tasks depending on the distance between the RSUs and vehicles to determine if they can communicate or not; this model is often referred to as the **unit disk model**. While this model is simplistic for application in the real-world, it is viable and could offer competitive performance. In the scope of our simulation scenario, we made several distance measuring simulations and got a **typical "transmission range" of 430 meters**, which is the distance at which RSUs received the packets with close to 0% loss. Since this is not a guaranteed transmission, the retransmission mechanism previously explained handles the possible packet losses.

5.2.2.4 Edge and Vehicle Capacities

There are two important aspects to consider when talking about Edge capacities:

- We define Edge capacity as storage for application data in an Edge. Vehicle capacity is also defined this way, but for storage in vehicles instead. This information refers to the amount of data that needs to be processed by an application's tasks. According to (95), the amount of data is one of the factors that sets the execution time of memory operations, while FLOPS define, among other variables, the execution time of computation. Both times combine to define the amount of time for an application to be processed. Also in the cited book, it is stated a metric Byte/FLOP which quantifies the amount of data a kernel requires to perform one FLOP. A vector processor delivers up to 1 Byte/FLOP and the NVIDIA Titan X used uses the Pascal micro-architecture for its Streaming Multiprocessor (96), which utilises

SIMD Vector Units. Under this assumption, we will assume that **storage, or capacity, is linearly proportional to FLOPS**;

- We use these capacities as a virtual computational cost of performing tasks. Due to cheap cost of storage, this metric allows to support a very large number of applications. Hence, for the purpose of our problem, we impose a limit to Edge capacities to a low value so that the benefits of our heuristic can be more clearly observed: if one Edge can, say, **hold 10 times the FLOPS of the application**, then at max, 10 tasks can be in that Edge's queue. This, if taken advantage by smart algorithms to distribute capacity more evenly, prevents edges from monopolising resources if we had wildly different processing speeds in all edges. This same approach was already proposed by some of the papers studied in 2.3.2. For the vehicle capacities, **we will define as 2 times the FLOPS of the application.**

5.2.2.5 Vehicle and Traffic Parameters

These values were introduced in Section 4.2.1.

- Vehicle Arrival Rate - We utilise the a SUMO script to generate routes, and in the script an option allows for to simulate the arrivals using a Binomial Distribution. But since the number of vehicles, N, is generally large (see Table 5.1) and the arrival period, p, is small (we used values below 0.05), we can approximate it to a Poisson Distribution instead. This approximation can be made if $N * p < 5$. The smallest used arrival rate is then $\frac{1}{0.05} = 20$ vehicles per second, which is a fair assumption in a busy street with multiple access points;
- Vehicle Maximum Speed - 50Km/h. This value was chosen based on the maximum speed allowed in Portuguese city streets;
- T_{start} - After being spawned in the Simulation, each vehicle follows an uniform distribution between 3 and 15 seconds to start requesting tasks;
- N_{tasks} - Each vehicle will request 3 tasks, which will make 3 seconds of images needed by the application. Each task is requested 1 second after the previous one;
- T_d - Task deadline is 1 second, as explained previously;
- C_v - This value is 5686.128 GFLOPS ($140 * 10e9FLOPS * 24FPS * 1.6923$);
- F_{limit} - The limit of refused tasks to deem the whole application unfeasible is 2;
- $T_{retrasnsmit}$ - 40 milliseconds per packet. This value has to be bigger than the sum of the Air Transmission and Propagation Delays, multiplied by 2, in order to know the RSU did not send an ACK;
- $T_{randomMA}$ - 50 milliseconds. This value derives from multiple runs of the Simulation, under heavy traffic, until an acceptable rate of success of 94% on a first transmission was reached, which does not affect significantly the performance of the system;

- Table 5.1 simply shows how we classify traffic by the number of the vehicles in the Simulation. These classifications are used when determining other values such as the Air Propagation Value explained before. These traffic ranges are used extensively as a main parameter, in the Results Section.

Classification	N vehicles
High Traffic	70-100
Medium Traffic	30-70
Low Traffic	0-30

Table 5.1: Traffic by number of vehicles.

5.2.2.6 RSU Parameters

These values were introduced in Section 4.2.3.

- R_{count} - This value is defined 5, meaning that the RSUs send the ACK message 5 times;
- $R_{retransmit}$ - This value defines the time between sending the R_{count} ACKS. It is defined as 200 milliseconds, since it is a bigger value than the sum of the Air Propagation and Transmission Delays.

5.2.2.7 Edge Parameters

These values were introduced in Section 4.2.2.

- α - Introduced in 4.1, it is a value used inside the Edge to represent the time unit at which the data gets processed. In this work, we consider $\alpha = 1ms$. This value defines the intervals of time the OMNeT Simulation has to simulate. Since it has to simulate each millisecond, each run takes a few seconds to complete, depending on the maximum Simulation Time;
- P_e - Taking into account that the GPU is capable of $10.97 * 10e12FLOPS$, the Edges process 10.97 GFLOPS of information each α.
- C_e - Since $C_v = 5686.128$ GFLOPS, and according to 5.2.2.4, the Edge capacities should be close to 55 000 GFLOPS. But we want to introduce some variability to the Edges. So we will consider that the Capacity of the Edges can vary between 40 000 to 70 000. This means that some of the Edges will have more capacity and therefore be more likely to be preferred by the system in the initial Solutions.

5.2.2.8 Simulated Annealing Values

These parameters where mentioned in Sections 3.5.1 and 4.3.1.

- Temperature - Defines the initial value of the algorithm. It is considered to be 100;

- β - Defines the end value of the algorithm. It is considered to be 1;
- Cooling Rate - Defines the rate at which the Equation 4.1 reaches β. The base value is 0.1, since in several comparison runs between both implemented heuristics, this value, or a higher one, is what allowed the Simulated Annealing to have approximately the same decision time as the Rank-Based Approach. This way, the results are based on performance of management alone;
- P(remaining) - this variable is compared against a value generated by an uniform distribution between 0 and 1, to decide if a new worst Solution remains to the next iteration.

5.2.2.9 Rank-Based Approach Values

These parameters where mentioned in Sections 3.5.2 and 4.3.2. These values were determined after having defined all other Simulation parameters and by running the Simulation with different values every time, until a small stable latency value was reached. They might not be the values that optimise the Heuristic, since the values themselves depend on road condition and the Edges' parameters, which are different every run.

- Base Merit - 0.75
- Merit Upper Limit - 0.9
- Merit Lower Limit - 0.4
- n_{setup} - 5
- x - This value corresponds to the random element in the Heuristic in the Key Utilisation Choice step. We use a uniform distribution between 0 and 1 to generate this value. It is then compared to the Merit parameters to determine if a new Learning Phase is needed.

5.3 Final Remarks

In this Chapter we detailed the Scenario considered in this thesis work, as well as presented and justified the relevant Scenario and Simulation parameters. Table 5.2 summarises all these parameters and the associated values, acting as a quick reference.

Parameter	Value
RSU/Edge Channel Propagation Delay	5.5us
RSU/Edge Channel Transmission Delay	2.68ms
Air Propagation Delay	Average: 240us
Air Transmission Delay	99.1ms
Routing Processing Delay	55us
Application Size	0.3344MB
Application Processing Time	528ms
Number of RSUs	6
Number of Edge Nodes	3
Maximum Vehicle Speed	50Km/h
Number of switches per link	1
Max Transmission Range	430 meters
Used txPower	20mW
Simulation Time	300s
Sumo Update Interval	1s
Manhattan Grid length and width	1000m
Vehicle Arrival Rate	20 vehicles per second
T_{start}	uniform(3s, 15s)
N_{tasks}	3
Task Deadline (T_d)	1s
Computation Required (C_v)	5686.128 GFLOPS
F_{limit}	2
$T_{retransmit}$	40ms per packet
$T_{randomMA}$	50ms
R_{count}	5
$R_{retransmit}$	200ms
α	1ms
P_e	120.97 GFLOPS
C_e	40 000-70 000
Temperature (initial)	100
Temperature (final = β)	1
Cooling rate (Simulated Annealing)	0.1
Base Merit (Rank-based Heuristic)	0.75
Upper Merit Limit (Rank-based Heuristic)	0.9
Lower Merit Limit (Rank-based Heuristic)	0.4

Table 5.2: Most Important Parameters and their Values.

Chapter 6

Evaluation of VEC Management Heuristics

In this Chapter, we will present the results obtained from all the simulations performed, in order to understand how the Heuristics and the system operate under different parameters. In a first perspective, we will only show the results that were obtained using the values explained in the previous Chapter. Then we will change some values and not only qualify the changes between Heuristics once more, but also compare the overall changes brought about by the variation in parameters.

6.1 Performance Metrics

This Section's performance metrics serve to quantitatively define the quality of the solutions developed in this thesis work. It is crucial that they are carefully chosen based on their relevance because the success or failure to correctly represent the System depends on if they accurately reflect what the system is capable of accomplishing.

- **Average Latency of Task Completion:** this metric represents the average end-to-end delay a vehicle experiences when requesting for edge service. It starts when a vehicle asks for a task to be completed and ends when it receives the processing results in their entirety. This value must be the smallest possible since many applications have strict time constraints to maintain the quality of service they require. This value only takes into consideration tasks completed before their deadline, because incomplete tasks have no end completion time and would erroneously skew the results negatively;

- **Service Ratio:** it is the number of tasks done on-time divided by the number of requested tasks (those done before the deadline and those done after). The better the management of resources, the better this ratio. It indicates how well can the heuristic operates based on the number of tasks to be serviced in a certain time interval. Of course, this value also depends on the established Edge capacities to deliver service: if we request way more service than the one installed, quickly this ratio worsens;

- **Average Edge Capacity Utilisation:** it indicates how often the Edge nodes resources are being utilised and to what extent. This measurement is done per Edge node;
- **Average Decision Time:** this is the average time that the resource management approaches developed in this thesis take to decide which task is sent to which Edge and which RSUs it passes through, after acquiring all the needed information to take such decision. It starts when a new task is requested by a vehicular user and ends when it reached a positive or negative outcome for that task (no service is offered if a decision cannot be made);

6.2 Statistical Procedures

6.2.1 Sampling of the Results

To obtain the results, these were the steps taken:

1. We gave each Performance Metric explained above the Signal mechanism employed by OMNeT++, to collect the information, as described in 4.4;
2. Then we performed 5 Simulation runs to each level of Traffic, per Heuristic. This number of runs, according to previous internal testings, allowed the system to give good representative values of the results and give them statistical significance. This means that a total of 30 runs took place. To each one of the runs, we altered the seed OMNeT uses to generate random numbers and we altered the routes of the vehicles in SUMO. The number of vehicles chosen was the maximum number in each level, meaning that for Low Traffic, we always used 30 vehicles, 70 to Medium Traffic and 100 to High Traffic, to see how the system reacted to the upper boundaries of the considered levels;
3. Then the values were exported to Python and processed, but not altered in any way. They were only used to calculate new values such as performing averages.

6.3 Results Demonstration and Discussion

6.3.1 Service Ratio

Table 6.1 shows the average Service Ratio for each Traffic Level and per Heuristic. The are several comparisons that can be made, which give us the following conclusions:

- In all Traffic Levels, the proposed Heuristic performs better than the baseline, if only slightly, meaning that it can service more requests if both have the same T_{start}. For example, for Low Traffic ($30 vehicles * 3 taskspervehicle = 90 tasks$), the Rank-Based Approach can meet the demands of 36 requests, while the Simulated Annealing can only service 33 tasks.

This is due to the fact that the proposed Heuristic can offer an improved assignment of the Edge resources, as it was expected since it uses past experiences to guide new decisions;

- The higher the number of vehicles, or in other words, the higher the number of tasks in the same amount of time, the lower the difference of results between Heuristics. This means that, at some point, the amount of tasks is so great that even good resource managing cannot account for the flood of requests with strict time constraints;
- With each level, both Heuristics lose about half of the capabilities of delivering service. For example, with 40 more cars, the Medium Traffic loses 19% to 21% of the Service Ratio, depending on Heuristic. This is an expected result: if a network is demanded more service without improving its own installed capacity (in this case processing speed since as we will see later, the processing time is the bottleneck) then the limitations are more apparent.

Traffic Level	Simulated Annealing	Rank-Based Approach
High Traffic (100)	10.7%	11.9%
Medium Traffic (70)	17.1%	19.1%
Low Traffic (30)	36.7%	40.2%

Table 6.1: Average Service Ratio with nominal parameters.

To further the discussion of these results, we now consider a secondary performance metric. There are two ways tasks can be considered as refused by the management system:

- the Orchestrator cannot reach any legal Solution and deems the task refused upfront, without even being sent to an RSU;
- the tasks was processed in an Edge, but when delivered back to the vehicle, the deadline was already met.

Table 6.2 shows us the **percentage of the former when divided against the total number of refused tasks**. As we can see, both Heuristics perform similarly. The vast majority of the refused tasks do not enter the Edge network, which is a good outcome since these tasks that are already estimated to fail do not use Edge resources that could go to tasks that could be completed on time. With the decrease in the number of vehicles in the Simulation, the amount of tasks that are refused upfront rises slightly. This happens because the Orchestrator has less information to process and so it can verify more accurately if a tasks will or not be completed once it enters the network.

Traffic Level	Simulated Annealing	Rank-Based Approach
High Traffic (100)	92.1%	93.66%
Medium Traffic (70)	94.65%	95.12%
Low Traffic (30)	97.33%	96.44%

Table 6.2: Tasks Refused Upfront with nominal parameters.

In the next Sections, we will return to the values of Table 6.1 and give them more meaning (i.e. we will discuss the absolute values obtained) since we first need a better insight of the other Performance Metrics which relate directly to this one.

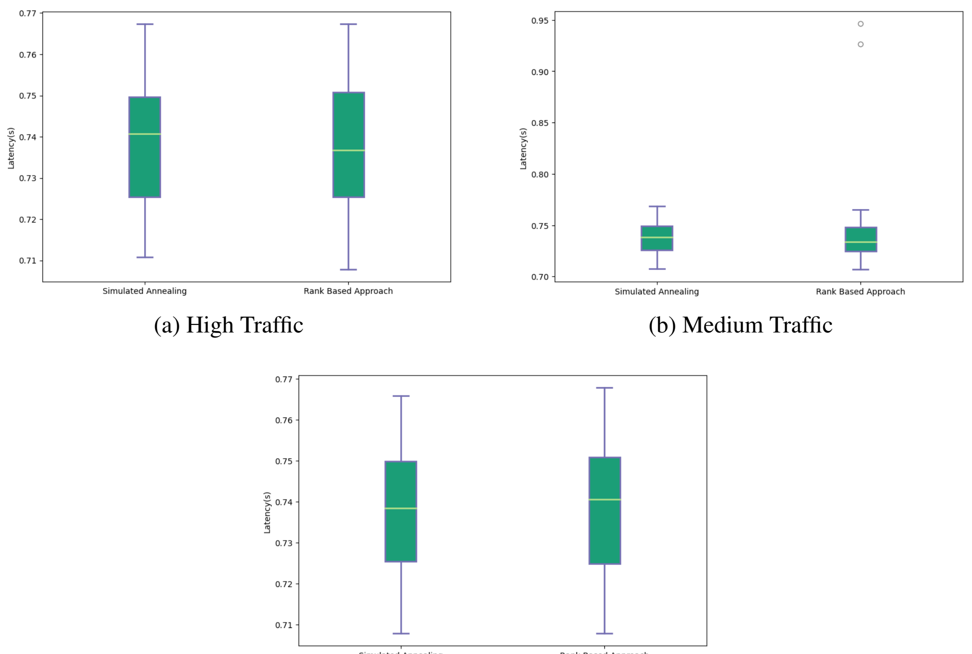
6.3.2 Average Latency

To demonstrate the results of this Metric, we chose two types of graphical representation:

- **Boxplot** - This type of plot is used to represent the distribution of values of all the latencies measured on all Traffic Levels, per Heuristic. The result can be seen in Figure 6.1. This plot was chosen because it allows us to perceive at a glance how the data is spread out and point out similarities or differences between data sets;
- **Time Series Graph** - When analysing how the average latency of different runs evolved over time, we quickly realised that, because the times of tasks' requests were different from one run to another, since T_{start} is random, calculating a mean of the results for identical points in time was a flawed endeavor. Even in the same Traffic Level and Heuristic, values changed from run to run. So we chose to represent all 5 running average latencies (i.e., an average that continually changes as more data points are collected) of a single Level in the same graph, and traced a doted line of the average of all points, independently of time, to identify a converge line. An example of the result can be seen in Figure 6.2 where we can see the performance of the Metric between both Heuristics, in High Traffic. More such graphs were made for all other Traffic levels.

Regarding the results of Figure 6.1:

- we can see that in all Traffic Scenarios, both Heuristics present very similar values of Average Latency of Task Completion, as evidenced by the shape of the boxes (that represent the second and third quartiles, from bottom to top respectively) and the median values (yellow line) which are less than 1ms apart. In High Traffic the proposed approach achieved a slightly lower median value, as opposed to the Low Traffic case, where the Simulated Annealing performed marginally better. The minimum and maximum values, represented by the blue whiskers, were also very similar, with less than half a millisecond in the worst case. These small discrepancies are explained due to the data sets we are comparing. These are to be expected if there are stochastic processes being simulated. If we used other samples, the best-performing Heuristic per Traffic Level could change but the variations would remain the same. This is know based on previous test runs during the development phase. The main conclusion of this comparison is that **although the Service Ratios differ, the time average latency resulting on the management of both Heuristics is approximately the same**;
- despite almost all of the points falling in between the quartiles, meaning that the variation is highly bounded, the small white dots depicted in the Medium Traffic boxplot tell us something important: in 2 occasions, the Ranked-Based Approach had to decide in a Solution



(a) High Traffic (b) Medium Traffic

(c) Low Traffic

Figure 6.1: Average Latency of Both Heuristics.

that would to take longer to complete, but that allowed to complete those two tasks in the first place. This behaviour did not happen in the Simulated Annealing Heuristic. The ability of the Rank Heuristic to adapt the Solution to better manage the Edge resources and prioritise task completion also explains **why the Service Ratio for this approach is higher than the other**. This behaviour is not due to this specific set of tests, it happens frequently;

- we need now to focus on the absolute values obtained, since they are the key to understanding why they are almost equal in all Scenarios. The deadline of all tasks was set to 1 second. This, and knowing that the tasks take are concentrated in a range between approximately 708 and 766 milliseconds, **leaves very little margin of maneuver for adding more than 2 tasks (from the same or from different vehicles) to any Edge** (since all Edges are equal) before a task is deemed as unfeasible with the resources available at the time. For example, if there are 2 tasks in an Edge, one of them with less than 200ms left to process and another still in the queue, then in a 1 second window, we know that those tasks will be delivered before the deadline. More tasks in that Edge are not able to do the same. It is important to note that the system supporting the Heuristics **prioritises completing tasks already decided** rather than taking on new ones with risk of not completing. This, in turn, **reduces the scope of available Solutions** and makes it more likely that both Heuristics (or any Heuristic for that matter) will present close results. The main influence of a tasks' latency is the processing time it requires on the Edge. For the specific application explained in the previous Chapter, it accounts for 528ms, or roughly 72% of the total latency. It is a **significant**

bottleneck of the system. If this value was smaller, due to smaller processing requirements or more powerful Edges, then the differences in Heuristics could be more easily grasped;

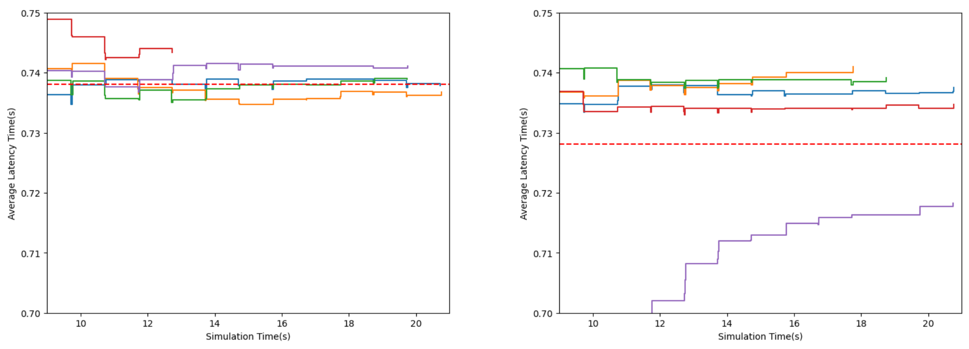
- when compared to the estimated latency of Equation 5.14 (732.161ms), we see that the estimate is almost equal to the average of the range of values obtained ($\frac{708+766}{2} = 737ms$). In fact, looking at the medians of the boxplots, this becomes even more apparent. This closeness of results tell us that the modelled values were well translated to the OMNeT++ entities' internal code. We can explain the small discrepancy partially based on the assumption made in Section 3.1, where we considered, for estimation purposes only, **that all Edge nodes are connected to a central Edge node, which in turn connects to all other Edges in the vicinity.** This simplification was what we used to only consider two allotments of the calculated Cabled Transmission and Propagation Delays, as well as the Routing Delay in the Equation. In reality, the Simulation may use more than 1 migration to deliver the task to the Edge and another to do the opposite. This makes the real time taken to a task to complete be higher than what was expected. Another possible cause for the higher values are the **retransmissions some tasks may need** to be received by the RSUs and vice-versa. As explained in 4.2.1.4, this adds more time to the total latency as the network is more utilised. The latency values smaller than the estimated latency can also be explained by the retransmissions, because the value used in 5.14 is an average. As such, some tasks may experience less retransmissions than the average and present a smaller total latency.

Finally we must discuss the Running Average Graphs of Figures 6.2, 6.3 and 6.4. In all Figures, we can see that the running averages remain relatively stable and even converge to a red doted line, which corresponds to the mean of the values used in each Figure. With this information, we can claim that over a sufficient amount of requests, the average latency can be predicted with accuracy. Some occasional phenomena can be seen, however. For example, in Figure 6.4 b), an average of a Rank-Based run (in green) has a sudden reduction in latency, likely from a set of tasks decided in a single Solution which was able to better manage the resources, due to a momentarily surplus of availability. In Figure 6.2 b) one average (in purple) starts off with a lower than normal latency and slowly converges. In Figure 6.3 b), one average (in orange) performs a bit worse than the majority in the beginning tasks, eventually converging at a fast rate. The experimentation and learning that occurs in the initial stages of the Rank-Based Heuristic explains this brief dichotomy of improved success or slightly worse results.

The Simulated Annealing Heuristic is more stable in its behaviour. Sometimes that is a good outcome, but in this context, some ability to innovate with Solutions can make the difference in the final Service Ratio. It is important to note that the convergence points correspond to the medians of the boxplots in Figure 6.1.

6.3.3 Average Decision Time

We chose to display the results of this time in boxplots and time series graphs, as previously done. Although the values presented in this Section are absolute ones, the analysis will only focus in



(a) Simulated Annealing (b) Rank-Based Approach

Figure 6.2: Running Averages of both Heuristics' Latency - High Traffic.

their relative terms as a way to compare the computational requirements of the Heuristics. This thesis work does not add this decision time as a component of the total latency, as it would in a real physical scenario. However, more powerful GPUs than the one considered in this work trivialise this time seeing as it already is a very small percentage of the overall latency observed. Figure 6.5 confirms this statement: the median is practically zero, since the computer from which these results were obtained is capable of producing them faster than the precision in the OMNeT statistics recording tools. Fortunately for this discussion, some values are not zero, and from those we were able to make running averages (although we will only show for High Traffic since the remaining follow the same behaviour), as depicted in Figure 6.6.

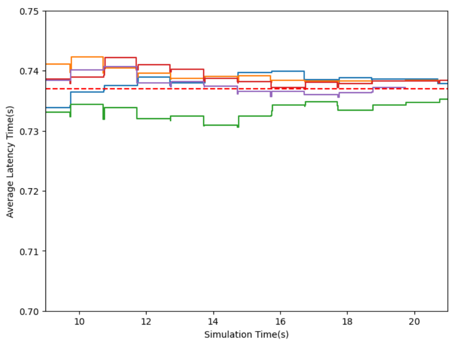
From the two Figures, we can conclude that:

- both Heuristics take practically the same amount of time to reach a decision, since the scattered points are near the same locations;
- there are many of such points that do not fall near the median, meaning that there is a high degree of variability in deciding the best Solution.

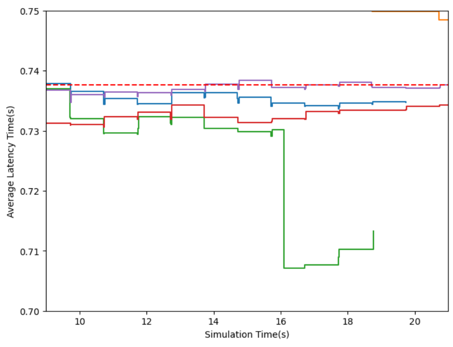
From these runs, we extracted the exact average times and we identified that the decision times for the Rank-Based Heuristic were nearly almost always faster (30us using a Intel Core i7-8750H @ 2.2GHz CPU to be precise).

6.3.4 Average Edge Capacity Utilisation

The higher or lower degree of utilisation of an Edge by the Heuristics depends, among other things, on the processing speed, P_e and the percentage of computation capacity available at the moment of decision, which equates to the "installed" capacity, C_e, minus the current utilisation. The more of these two parameters an Edge has, the more likely is to be chosen as a good candidate for a task placement in a Solution. Since P_e was assumed to be the same for all 3 edges, C_e is the only factor that introduces variability. The installed Edge capacity is a parameter that can vary from 40 000 to 70 000: as explained in 5.2.2.7, these values are an arbitrary multiple value between

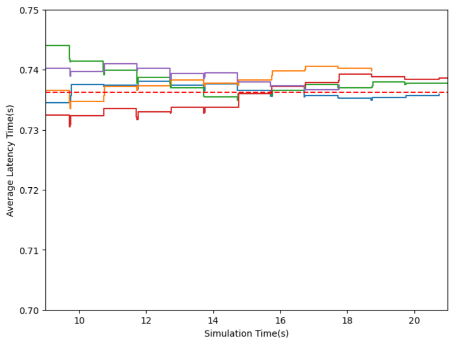


(a) Simulated Annealing

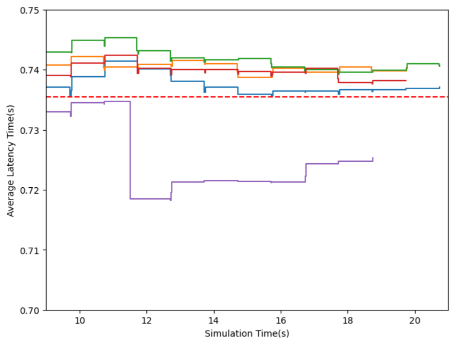


(b) Rank-Based Approach

Figure 6.3: Running Averages of both Heuristics' Latency - Medium Traffic.



(a) Simulated Annealing



(b) Rank-Based Approach

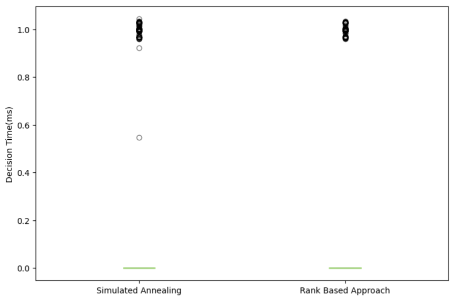
Figure 6.4: Running Averages of both Heuristics' Latency - Low Traffic.

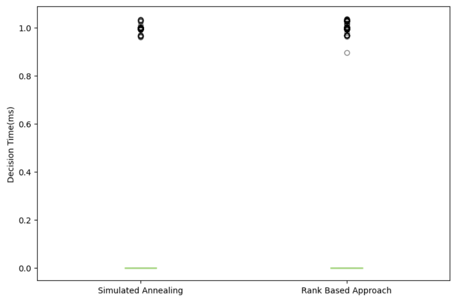
approximately 70% and 130% of the nominal application size times 10. To make the results more meaningful, we decided to maintain the same capacities for each run, since we are only interested in the percentage differences. The actual value just has to be high enough as to leave room to tasks to be placed, because an Edge is chosen based on percentage available and not the absolute value. In a first thought, the reader might think that this detail might make both Heuristics tend to choose the same Edge. This would be correct, if not counting with the internal parameters of the Heuristics that also affect the choice. To better comprehend this idea we will now analyse the results as they appear in Table 6.3. Note that the amount of capacity considered is also indicated next to the Edge number. These values were taken from the initial sample of 30 runs.

Edge	Simulated Annealing	Rank-Based Approach
1(55k)	10.3%	12.68%
2(40k)	14.2%	17.6%
3(70k)	8.1%	9.9%

Table 6.3: Average Edge Capacity Utilisation per Edge.

The percentages of the Simulated Annealing runs correspond exactly to the amount of what


(a) High Traffic


(b) Medium Traffic

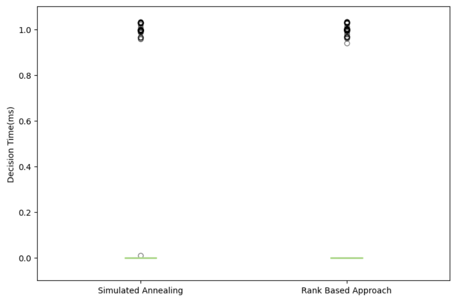
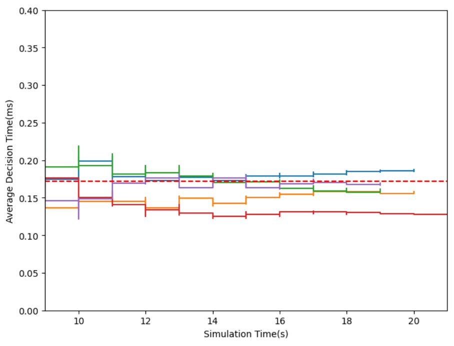

(c) Low Traffic

Figure 6.5: Average Decision Time of Both Heuristics.


(a) Simulated Annealing

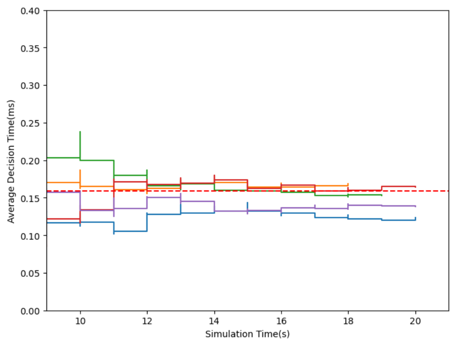

(b) Rank-Based Approach

Figure 6.6: Running Averages of both Heuristics' Decision Time - High Traffic.

one task in each of the Edges would occupy, when not yet processed. Since this is an average, this means that at most, 2 tasks, in different degrees of completion, can be put in an Edge by this Heuristic. One important takeaway is that the Edges are utilised equally and this also happens with the Rank-Based Approach. However, the percentages of utilisation are slightly higher in the latter, more specifically by roughly 25% when compared to the Simulated Annealing. This amounts to almost using an optimum degree of the remaining processing needs of the 2 tasks that can be in each Edge at one time.

It is important to clarify that both Heuristics try to balance the load between Edges, and this was an expected outcome, since the tasks required computational power and the Edges' processing speeds are the same. What is a fact is that the Rank-Based Approach can utilise more of what the Edges provide, even in a bottleneck scenario as this one. In theory, we had reason to believe that the proposed Approach could more efficiently utilise the resources available, because the Learning Phase and the remaking of Keys can reach more and possibly more varied Solutions in the long run. That was seen to be a reality, in practice, even if the values between Heuristics are not greatly expressive.

6.4 Design-Space Exploration

The purpose of this Section is to alter specific parameters and observe, analyse and understand the results obtained, and how they deviate from the results studied so far. Firstly, we will modify the Scenario parameter T_{start} and then we will conduct a study of a different application altogether.

6.4.1 Details on Modifiable Parameters

Many of the parameters explained in 5.2 are required to be immutable throughout the simulations, to preserve the realistic approach we are trying to give to this thesis work. Other parameters such as the Traffic Level boundaries or the internal parameters of each Heuristic can be changed, but we will refrain from doing so for the remaining of this thesis, since the nominal values presented are already good enough to use and expect good representative results, as measured and concluded in prior testings of the system, during development. We are left with two types of parameters that are not critical to the realism of the simulation, but are significant enough to change some of the the results: **Scenario Parameters and Application Parameters**.

1. Scenario Parameters

   - T_{start} - This value, nominally, is a random uniform between 3 and 15. The lower value must be fixed because the OMNeT Simulation needs a small setup time to connect SUMO, and, if this value was lower, some complications could arise when preparing tasks of vehicles that are not fully initialised. The upper value, however, is modifiable at will. We chose 15 because the derived 12 second window where all vehicles start their tasks is small, when also considering the next modifiable value. This puts the system in a small strain that we want to force to test and showcase some limits.

2. Application Parameters

   - N_{tasks} - This value was chosen to be 3, meaning that each second the vehicle sends a new task. Although, by itself, it may not seem many requests from a vehicular application, if we consider that, with 30 or more vehicles sending their first request within the 12 second window referred above and needing more every second, the effort required by the 3 Edge nodes to service all tasks is significant and that allows us to better perceive how well or not the Heuristics manage the resources. Additionally, 3 seconds of processed information to be used by an Image Processing software, in a high-speed vehicular scenario, is more than sufficient for it to work properly;

- F_{limit} - This value was chosen to be 2, meaning that only 1 task per vehicular application is needed to function. This requirement is indeed lenient, but we wanted to distinguish this specific application from Safety Critical ones, where even 1 missed task can jeopardise the correct functioning of the application;
- Application Processing Time - Introduced in 5.2.1.7, this value is, nominally, $528ms$. This time is dependent on the application's required FLOPS and in the Edge processing speed. In this thesis, we will analyse the impact of a reduction in the former, by choosing an alternative, less computationally demanding, application.

6.4.2 Modified T_{start}

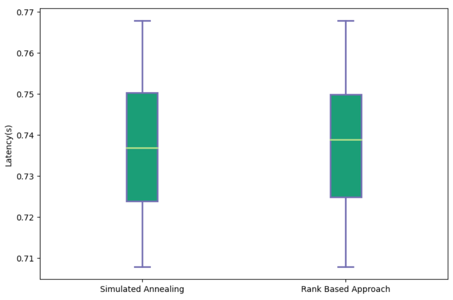
We increase the T_{start} upper boundary from 15 to 30 seconds (and perform runs with different seeds and vehicle routes, as done before). This allows for more tasks to be serviced since we have the same number of requests over a bigger time window. Table 6.4 justifies this conclusion by showing us increases in Service Ratio across the board. The values obtained for this table and for the following plots are for Low and High Traffic only, since Medium Traffic lies in the middle of the results of the other two. Compared to the nominal Low Traffic values, the increase of successfully serviced tasks rose by more than 30% for both Heuristics. For High Traffic, the rise was only by 14-16% but still significant. The difference between Heuristics remained approximately the same for Low Traffic but increased for the High Traffic scenario from 1.2% to almost 4%. Both Heuristics have more time to service the current requests before more arrive, so it is natural this Performance Metric would go up.

Traffic Level	Simulated Annealing	Rank-Based Approach
High Traffic	24.8%	28.3%
Low Traffic	69.0%	73.8%

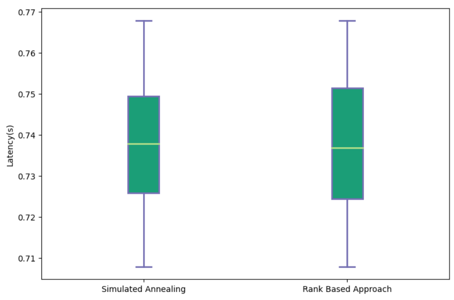
Table 6.4: Service Ratio with modified T_{start}.

Now analysing the Average Latency in the boxplots of Figure 6.7, we can perceive that the values remain practically the same as they were with the nominal T_{start}. Only successful tasks count toward the latency average of the Heuristics, so if the bottleneck created by the high application processing delay in the Edges persists, then the task latencies will remain unaltered. This also means the differences between Heuristics are kept to almost the same.

Regarding the running averages for both High and Low Traffic, in Figures 6.8 and 6.9 respectively, the same analysis can be done to these plots as it was done in the nominal case. The main difference is that there is a more pronounced transient response in the initial stages of the averages' calculation: higher maximums and lower minimums, but ultimately they converge to the same stable value we discussed before.

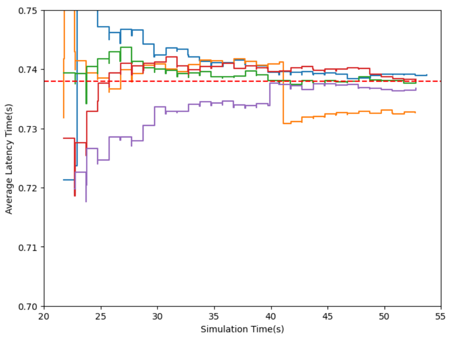


(a) High Traffic

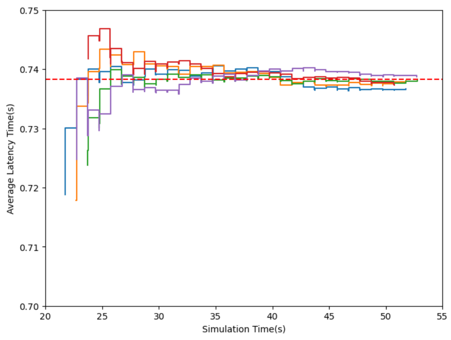


(b) Low Traffic

Figure 6.7: Average Latency of Both Heuristics with modified T_{start}.

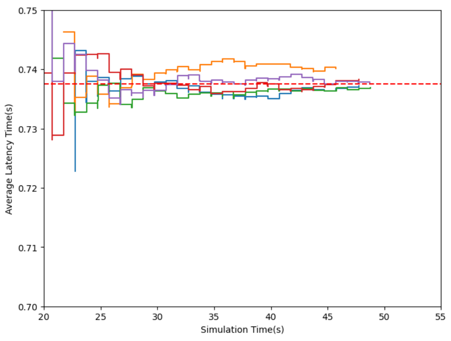


(a) Simulated Annealing

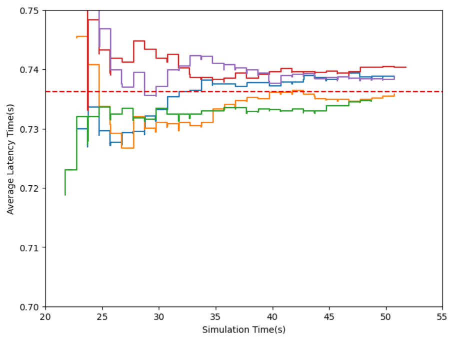


(b) Rank-Based Approach

Figure 6.8: Running Averages of both Heuristics' Latency with modified T_{start} - High Traffic.



(a) Simulated Annealing



(b) Rank-Based Approach

Figure 6.9: Running Averages of both Heuristics' Latency with modified T_{start} - Low Traffic.

Finally, is it important to say that the Edge Capacity Utilisation presented approximately the same values as the nominal scenario.

6.4.3 Analysis of an Alternative Application

The previous application, as it was explained, heavily influenced the results obtained. More importantly, the differences between Heuristics were not very apparent. To provide a better view of what the Ranked-Based Approach is capable of, in this Section, we will explore another design consideration that will expand the range of results we can obtain: **we will consider an application that needs only one fifth of the processing requirements of the nominal application**. This means that the processing time will be cut to one fifth as well, and suddenly, instead of 528ms of processing time, we will only need approximately 100ms, which, according to Table 2.2, could be considered a Safety Application. We understand that this assumption is not backed by a real study of an application that meets this requirement, as it was done in 5.2.1.1 for the nominal application, but it is an important enough design consideration to test the boundaries of the management system.

As well as the change described above, the Edge capacities will be scaled down to continue to be able to hold 10 tasks at maximum, as assumed in 5.2.2.4. We will only use the Low and High Traffic Levels, as the Medium Level presents medium values and is not critical for the understanding of the results. Finally, for the sake of brevity, we will alter all the modifiable parameters at the same time to offer a more straightforward understanding of the benefits, unlike previous Sections in which we only varied one parameter at a time for more accurate comparisons. The progression of these changes will be to put more strain in the system with each test.

6.4.3.1 Service Ratio

Config.	Traffic Level	N_{tasks}	F_{limit}	T_{start}	Simulated Annealing	Rank-Based Approach
1	Low (30)	5	3	3,10	83.3%	93.4%
2	Low (30)	7	4	3,8	83.3%	90%
3	Low (30)	7	1	3,5	73.3%	88.1%
4	High (100)	3	2	3,10	66.7%	72.2%
5	High (100)	6	4	3,6	32.2%	35.5%

Table 6.5: Service Ratios for an alternative vehicular application.

Table 6.5 shows us the Service Ratios obtained in **five test configurations**. Each configuration followed the same sampling of the results described in 6.2.1. Compared to Tables 6.1 and 6.4, the difference between the Heuristics is clearer: the proposed Heuristics performs better than the Simulated Annealing, especially when the Traffic Level is Low. Rank-Based was able to handle 15% more tasks than SA (in the third test). In the last test configuration, when the requirements where the most strict, the Service Ratio was closer to the results obtained in earlier Sections. A less demanding application allows for an increase in the range of Solutions the Heuristics can reach. This is a scenario in which the Rank-Based Approach presents a more efficient procurement of where the Edge resources would be best utilised.

6.4.3.2 Average Latency

This performance metric was the most affected by the change in application, and where the difference between Heuristics can be more easily grasped. With the reduction of processing requirements, the estimated Round Trip Time of tasks reduced to only 304.161ms, which allows up to 4 or 5 tasks in an Edge, instead of the previously 2 or 3.

In Figures 6.10 and 6.11 we can see the results of the Latencies obtained in two of the five tests described previously. We will only exhibit these two configurations, because they sufficiently convey the two new main differences that are also present across the remaining:

- the range of latencies is much wider. In the 1st test, the majority falls between 380ms to 610ms. In the 4th test, the upper values of the 3rd quartile wide to up to 750ms, in the Simulated Annealing, and the lower values of the 2nd quartile are only slightly above 400ms. Both runs present the expected increase of Solution diversity. Additionally, as the whiskers and outliers show, it can happen that the latency of a task completion can be close to the deadline or almost as fast as theoretically estimated, due to the increased liberty the Heuristics have in searching through the range of Solutions;

- the median values of the Heuristics also differ more substantially: in the 1st test, the Rank-Based Approach showed a value of 487.3ms, almost 27ms faster on average than the Simulated Annealing, which got 510.1ms. In the 4th test, the difference was smaller: the proposed Heuristic was 20ms faster, with a median of 547.9ms. As the Runs' requirements increased, by increasing the amount of expected tasks per unit of time or by decreasing the F_{limit} compared to N_{tasks}, the difference in median values diminished, but it still remained of at least 10ms.

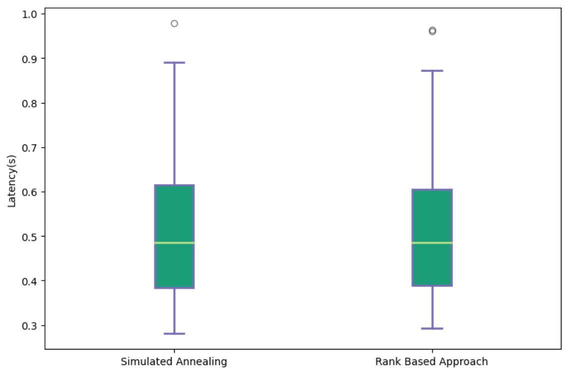


Figure 6.10: Average Latency of Both Heuristics - 1st Test Configuration.

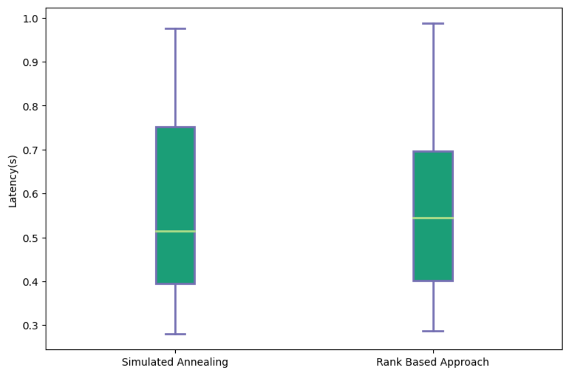


Figure 6.11: Average Latency of Both Heuristics - 4th Test Configuration.

6.4.3.3 Average Edge Capacity Utilisation

To finish this discussion, it is important to mention that the Average Edge Utilisation also changed, as it can be seen in Figure 6.6. Compared to the previous results, the values acquired are roughly twice as higher. This was also an expected outcome, since the reduction in application size means tasks have more time they can spend in the Edges' queue, without it compromising the delivery before the deadline. The application processing delay was reduced by one fifth of the original, but all other delays remained the same, so the increase could never directly proportional to the decrease in the processing delay alone. The differences in the Heuristics are more pronounced, with the Rank-Based Heuristic being better than the Simulated Annealing Heuristic by a wider margin than the values of Table 6.3.

Edge	Simulated Annealing	Rank-Based Approach
1(10k)	20.4%	25.2%
2(8.5k)	27.9%	34.3%
3(11.5k)	16.2%	19.7%

Table 6.6: Average Edge Capacity Utilisation per Edge - Alternative application.

6.5 Summary & Additional Discussion

We present a summary of this Chapter's most important results:

- with nominal values, the proposed Heuristic performs slightly better than the Simulated Annealing, when comparing the Service Ratio and Edge Utilisation Capacity performance metrics. The Average Latency is very similar between the two. This can be explained due to the high processing time the chosen application has when considering that the deadline for all tasks is 1 second. This puts a bottleneck on the amount of tasks that can be serviced on time and on the range of Solutions available to be analysed by the Heuristics;

- when T_{start} is increased, the amount of serviced tasks that can be serviced also increases, since the additional time between tasks of different vehicles leaves more time to the existing tasks to be processed with the limited Edge resources available. The latencies remain the same, since the application itself did not change;
- when the vehicular application's processing requirements are substantially reduced, the differences between Heuristics increase. The proposed Heuristic's advantages are more easily portrayed since the scope of available Solutions to work with is much larger. Additionally, the performance metrics fare better across the board.

Regarding this last point, throughout Section 6.4.3, the results obtained evidence the fact that the proposed Heuristic performs significantly better than the baseline approach, when the application requirements are less demanding. We believe, according to the internal information available and processed during our implementation phase, that this behaviour can be justified by the way the Learning Phase and the need of Key recalculation is considered. The Learning Phase tries to put the task in the Edges and calculates the *makespan* almost in a brute force manner: it stops before the decision calculation can be considered significant for delay purposes. This assures a wide search over many of the possible legal Solutions available. This process would take too long, if it was used at every new task. The use of the Merit System and the probabilistic factors of Key recalculation, explained in Section 4.3.2, limit the number of Learning Phases to the minimum required for the Heuristic to perform well with past results. The Simulated Annealing Heuristic does not guarantee that as much of the scope of Solutions is analysed, since the *cooling rate* was limited to equalise the two approaches' decision time, as it was seen in 6.3.3. This lack of previous data means that the baseline approach will always have to recalculate with each new task request and will be unable to reach better Solutions each time.

Even though the results, when using the nominal application, were not very expressive of the differences in Heuristics, by further exploring the design-space (using a different vehicular application) it showed that the proposed Rank-Based Approach has potential to shine when processing times are low, when compared to the total latency.

Chapter 7

Conclusion

This thesis aimed to understand the resource assignment problem prevalent in Vehicular Edge Computing scenarios and identify and study a novel solution that could effectively provision a large number of vehicular users, with the goal of reducing the total end-to-end latency. The method proposed was a Rank-Based Heuristic which, when compared to the Simulated Annealing Heuristic, was able to achieve (in the initial scenario) a higher number of successfully serviced vehicles (up to 3.5%), utilise more extensively the available Edge resources (up to 3.4%) and reach an assignment decision in a faster manner. Additionally, based on a quantitative analysis of the average latency of both Heuristics, using two vehicular applications with varying processing requirements, it can be concluded that the proposed Heuristic achieves smaller end-to-end latencies, even in bottleneck scenarios as the one first considered in Chapter 5.

After a thorough review of the current State of the Art, in Chapter 2, we concluded that there was a gap in knowledge in the way traditional solutions approach the research problem: no previous academic work had been made that tried to incorporate the concepts of "scoring" and utilising previous assignment decisions into an Edge Resource Management point of view. A few works did use historical information that enabled some part of their algorithms to reach better outcomes, but they do not fully store or adapt accumulated information to help guide future resource requests from the users. With the proposed Rank-Based Approach, we expected that, by finding a way to accurately represent the results of previously made assignments by the Heuristic, we could utilise to full potential the benefits of knowing near optimal decisions and using them again when network conditions would be similar. This is in clear contrast to only searching through a limited part of the possible range of solutions and picking the best one, as the implemented Simulated Annealing and many other academic works do. The results match our expectations, since all the performance metrics, chosen to provide a clear view of the gathered statistical information, showed that the proposed Heuristic was able to reach solutions with reduced latency in a consistent fashion.

While the main vehicular application chosen for this work did greatly influence the scope of the assignment solutions that could be obtained from both Heuristics, which in turn made the differences between them not very apparent, the research clearly illustrate that having even some sort of historical information to help draw conclusions, makes for a better, even if slight, overall performance. This raises the question of whether this increase in algorithmic complexity is worth the small latency gains, when considering strict scenarios as the chosen one for this thesis. Nowadays, with the immensely powerful processing units commercially available, which are able to handle great amounts of calculations per second, the computation overhead is negligible and so can be the complexity of the Heuristics.

It is important to remember that we also tested the system with a revised application which more definitively showed the improvement the proposed Heuristic is capable: in the initial stages of this thesis, we started looking for an application and values to use which conveyed the proper realism to this work; we implemented it in the simulation system in the most accurate way possible; finally we realised that the advantages were not being demonstrated as well as they could. Section 6.4.3 is the result of a battery of tests which could be expanded upon in future work. This realisation is one of the lessons learned from this thesis. It is possible that a better work methodology was to improve on the selection of the application, and then adapt the implementation from it. On the other hand, this application, which would showcase the differences in a more expressive way, not only could have taken more time to fully justify its use, but it could also had provided values which were not close of being a real representation of reality.

This thesis utilised the following work flow:

1. Reviewed the State of the Art and found an area to which we thought we could offer a contribution;

2. Studied the problem and proposed a new approach to tackle it - the Rank-Based Approach;

3. Designed a specific scenario to test the Heuristic;

4. Studied the best simulation and mobility tools to implement the scenario in which we would compare the approach to the baseline one. The chosen tools were OMNeT++, Veins and SUMO;

5. Found the limitations of the tools and implemented mechanisms to emulate other commonplace considerations. Examples of this work are the Mobility Trace Generation procedure, the Retransmission Mechanism and the Historical Latency Information for RSU Selection procedure;

6. Focused on obtaining realistic values for all the simulation parameters to be included, such as the typical communication delays in wireless and cabled transmissions or a real vehicular application in which the Heuristics could apply their capabilities;

7. Ran tests according to a specific methodology to obtain statistical significance and analysed the resulting information to conclude about the Heuristics' performance.

To better understand the potential of the Rank-Based Heuristic has, future studies could address the fine-tuning of its internal parameters, as to find a configuration which could possibly achieve smaller latencies. This thesis work did try to find acceptable values for obtaining expressive results, but room still exists for improvement. As it was said before, one other suggestion is to try different types of applications that, without losing realistic value, can better show the full capabilities the proposed Heuristic. Another suggestion is to use the Simulation environment developed for this work to develop new or other known resource assignment approaches. This can lead to more direct comparisons between the proposed solution and future works on the same subject. Further research is needed to determine the viability of this solution when traffic conditions are even more accentuated. The maximum number of vehicles we tested the simulations with was 100, since the simulation tool, in the Personal Computer used to develop this work, was already struggling to run the tests. With more computational resource available, practitioners could try and increase the number of vehicles to the thousands, and even expand the scenario's distance boundaries.

References

[1] L. Liu, C. Chen, Q. Pei, S. Maharjan, and Y. Zhang, "Vehicular edge computing and networking: A survey," *arXiv preprint arXiv:1908.06849*, 2019.

[2] S. Raza, S. Wang, M. Ahmed, and M. R. Anwar, "A survey on vehicular edge computing: Architecture, applications, technical issues, and future directions," *Wireless Communications and Mobile Computing*, vol. 2019, 2019.

[3] C. Sommer, R. German, and F. Dressler, "Bidirectionally coupled network and road traffic simulation for improved ivc analysis," *IEEE Transactions on mobile computing*, vol. 10, no. 1, pp. 3–15, 2010.

[4] Gartner Inc., "Gartner's report." `https://www.gartner.com/en/newsroom/press-releases/2015-01-26-gartner-says-by-2020-a-quarter-billion-connected-vehicles-will-enable-new-in-vehicle-services-and-automated-driving-capabilities`, Last accessed on 2019-11-28.

[5] Intel, "Technology and requirements for self-driving cars," 2008. Available at `https://www.intel.com/content/www/us/en/automotive/driving-safety-advanced-driver-assistance-systems-self-driving-technology-paper.html`, last accessed on the 27th of November, 2019.

[6] L. Abdi, F. B. Abdallah, and A. Meddeb, "In-vehicle augmented reality traffic information system: a new type of communication between driver and vehicle," *Procedia Computer Science*, vol. 73, pp. 242–249, 2015.

[7] A. Frotscher and T. Scheider, "Coopers project: Development of an its architecture for cooperative systems on motorways," in *15th World Congress on Intelligent Transport Systems and ITS America's 2008 Annual Meeting*, 2008.

[8] E. Ahmed and H. Gharavi, "Cooperative vehicular networking: A survey," *IEEE Transactions on Intelligent Transportation Systems*, vol. 19, no. 3, pp. 996–1014, 2018.

[9] Cisco, "Edge computing and 5g." Available at `https://www.cisco.com/c/en/us/solutions/enterprise-networks/edge-computing-architecture-5g.html`, last accessed on the 9th of February, 2020.

[10] R. Tomar, M. Prateek, and G. Sastry, "Vehicular adhoc network (vanet)-an introduction," 2016.

[11] H. Peng, L. Liang, X. Shen, and G. Y. Li, "Vehicular communications: A network layer perspective," *IEEE Transactions on Vehicular Technology*, vol. 68, no. 2, pp. 1064–1078, 2018.

[12] H. A. El Zouka, “A secure interactive architecture for vehicular cloud environment,” in *2016 IEEE International Conference on Smart Cloud (SmartCloud)*, pp. 254–261, IEEE, 2016.

[13] M. N. Birje, P. S. Challagidad, R. Goudar, and M. T. Tapale, “Cloud computing review: concepts, technology, challenges and security,” *International Journal of Cloud Computing*, vol. 6, no. 1, pp. 32–57, 2017.

[14] W. Song and X. Su, “Review of mobile cloud computing,” in *2011 IEEE 3rd International Conference on Communication Software and Networks*, pp. 1–4, IEEE, 2011.

[15] M. R. Rahimi, J. Ren, C. H. Liu, A. V. Vasilakos, and N. Venkatasubramanian, “Mobile cloud computing: A survey, state of art and future directions,” *Mobile Networks and Applications*, vol. 19, no. 2, pp. 133–143, 2014.

[16] P. Mach and Z. Becvar, “Mobile edge computing: A survey on architecture and computation offloading,” *IEEE Communications Surveys & Tutorials*, vol. 19, no. 3, pp. 1628–1656, 2017.

[17] Y. C. Hu, M. Patel, D. Sabella, N. Sprecher, and V. Young, “Mobile edge computing—a key technology towards 5g,” *ETSI white paper*, vol. 11, no. 11, pp. 1–16, 2015.

[18] N. Abbas, Y. Zhang, A. Taherkordi, and T. Skeie, “Mobile edge computing: A survey,” *IEEE Internet of Things Journal*, vol. 5, no. 1, pp. 450–465, 2017.

[19] M. Chiang and T. Zhang, “Fog and iot: An overview of research opportunities,” *IEEE Internet of Things Journal*, vol. 3, no. 6, pp. 854–864, 2016.

[20] A. V. Dastjerdi and R. Buyya, “Fog computing: Helping the internet of things realize its potential,” *Computer*, vol. 49, no. 8, pp. 112–116, 2016.

[21] D. Talia, “Clouds for scalable big data analytics,” *Computer*, no. 5, pp. 98–101, 2013.

[22] X. Xu, Y. Xue, X. Li, L. Qi, and S. Wan, “A computation offloading method for edge computing with vehicle-to-everything,” *IEEE Access*, vol. 7, pp. 131068–131077, 2019.

[23] I. Ku, Y. Lu, M. Gerla, R. L. Gomes, F. Ongaro, and E. Cerqueira, “Towards software-defined vanet: Architecture and services,” in *2014 13th annual Mediterranean ad hoc networking workshop (MED-HOC-NET)*, pp. 103–110, IEEE, 2014.

[24] A. M. Vegni, M. Biagi, R. Cusani, *et al.*, “Smart vehicles, technologies and main applications in vehicular ad hoc networks,” *Vehicular technologies-deployment and applications*, pp. 3–20, 2013.

[25] K. Lee, J. Kim, Y. Park, H. Wang, and D. Hong, “Latency of cellular-based v2x: Perspectives on tti-proportional latency and tti-independent latency,” *IEEE Access*, vol. 5, pp. 15800–15809, 2017.

[26] H. Vahdat-Nejad, A. Ramazani, T. Mohammadi, and W. Mansoor, “A survey on context-aware vehicular network applications,” *Vehicular Communications*, vol. 3, pp. 43–57, 2016.

[27] I.-H. Bae and S. Olariu, “A tolerant context-aware driver assistance system for vanets-based smart cars,” in *2010 IEEE Global Telecommunications Conference GLOBECOM 2010*, pp. 1–5, IEEE, 2010.

[28] W. Woerndl and R. Eigner, “Collaborative, context-aware applications for inter-networked cars,” in *16th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE 2007)*, pp. 180–185, IEEE, 2007.

[29] S. Umamaheswari and R. Priya, “An efficient healthcare monitoring system in vehicular ad hoc networks,” *International Journal of Computer Applications*, vol. 78, no. 7, 2013.

[30] P. Kamal, R. Raw, N. Singh, S. Kumar, and A. Kumar, “Vanet based health monitoring through wireless body sensor network,” in *2016 3rd International Conference on Computing for Sustainable Global Development (INDIACom)*, pp. 2865–2871, IEEE, 2016.

[31] C. Chen, T. Qiu, J. Hu, Z. Ren, Y. Zhou, and A. K. Sangaiah, “A congestion avoidance game for information exchange on intersections in heterogeneous vehicular networks,” *Journal of Network and Computer Applications*, vol. 85, pp. 116–126, 2017.

[32] W. Alghamdi, “Improving driver’s behavior using context-aware systems,” *Procedia Computer Science*, vol. 10, pp. 1213–1216, 2012.

[33] Q. Yuan, H. Zhou, J. Li, Z. Liu, F. Yang, and X. S. Shen, “Toward efficient content delivery for automated driving services: An edge computing solution,” *IEEE Network*, vol. 32, no. 1, pp. 80–86, 2018.

[34] H. Hisamatsu, G. Hasegawa, and M. Murata, “Non bandwidth-intrusive video streaming over tcp,” in *2011 Eighth International Conference on Information Technology: New Generations*, pp. 78–83, IEEE, 2011.

[35] Z. Chen, Q. He, Z. Mao, H.-M. Chung, and S. Maharjan, “A study on the characteristics of douyin short videos and implications for edge caching,” *arXiv preprint arXiv:1903.12399*, 2019.

[36] P. Zhou, W. Zhang, T. Braud, P. Hui, and J. Kangasharju, “Enhanced augmented reality applications in vehicle-to-edge networks,” in *2019 22nd Conference on Innovation in Clouds, Internet and Networks and Workshops (ICIN)*, pp. 167–174, IEEE, 2019.

[37] H. El-Sayed, S. Sankar, M. Prasad, D. Puthal, A. Gupta, M. Mohanty, and C.-T. Lin, “Edge of things: the big picture on the integration of edge, iot and the cloud in a distributed computing environment,” *IEEE Access*, vol. 6, pp. 1706–1717, 2017.

[38] M. Narasimha, V. Desai, G. Calcev, W. Xiao, P. Sartori, and A. Soong, “Performance analysis of vehicle platooning using a cellular network,” in *2017 IEEE 86th Vehicular Technology Conference (VTC-Fall)*, pp. 1–6, IEEE, 2017.

[39] R.-H. Huang, B.-J. Chang, Y.-L. Tsai, and Y.-H. Liang, “Mobile edge computing-based vehicular cloud of cooperative adaptive driving for platooning autonomous self driving,” in *2017 IEEE 7th International Symposium on Cloud and Service Computing (SC2)*, pp. 32–39, IEEE, 2017.

[40] A. Kovalenko, R. F. Hussain, O. Semiari, and M. A. Salehi, “Robust Resource Allocation Using Edge Computing for Vehicle to Infrastructure (V2I) Networks,” in *2019 IEEE 3rd International Conference on Fog and Edge Computing (ICFEC)*, pp. 1–6, IEEE, may 2019.

[41] S. Choo, J. Kim, and S. Pack, “Optimal task offloading and resource allocation in software-defined vehicular edge computing,” in *2018 International Conference on Information and Communication Technology Convergence (ICTC)*, pp. 251–256, IEEE, 2018.

[42] J. Li, C. Natalino, D. P. Van, L. Wosinska, and J. Chen, "Resource management in fog-enhanced radio access network to support real-time vehicular services," in *2017 IEEE 1st International Conference on Fog and Edge Computing (ICFEC)*, pp. 68–74, IEEE, 2017.

[43] M. Li, F. R. Yu, P. Si, H. Yao, and Y. Zhang, "Software-defined vehicular networks with caching and computing for delay-tolerant data traffic," in *2018 IEEE International Conference on Communications (ICC)*, pp. 1–6, IEEE, 2018.

[44] Z. Miao, C. Li, L. Zhu, X. Han, M. Wang, X. Cai, Z. Liu, and L. Xiong, "On resource management in vehicular ad hoc networks: A fuzzy optimization scheme," in *2016 IEEE 83rd Vehicular Technology Conference (VTC Spring)*, pp. 1–5, IEEE, 2016.

[45] R. Yu, J. Ding, X. Huang, M.-T. Zhou, S. Gjessing, and Y. Zhang, "Optimal resource sharing in 5g-enabled vehicular networks: A matrix game approach," *IEEE Transactions on Vehicular Technology*, vol. 65, no. 10, pp. 7844–7856, 2016.

[46] X. He, Z. Ren, C. Shi, and J. Fang, "A novel load balancing strategy of software-defined cloud/fog networking in the internet of vehicles," *China Communications*, vol. 13, no. 2, pp. 140–149, 2016.

[47] M. M. K. Tareq, O. Semiari, M. A. Salehi, and W. Saad, "Ultra reliable, low latency vehicle-to-infrastructure wireless communications with edge computing," in *2018 IEEE Global Communications Conference (GLOBECOM)*, pp. 1–7, IEEE, 2018.

[48] Y. Zhang, J. Zhao, and G. Cao, "On scheduling vehicle-roadside data access," in *Proceedings of the fourth ACM international workshop on Vehicular ad hoc networks*, pp. 9–18, 2007.

[49] S. Wu, W. Xia, W. Cui, Q. Chao, Z. Lan, F. Yan, and L. Shen, "An efficient offloading algorithm based on support vector machine for mobile edge computing in vehicular networks," in *2018 10th International Conference on Wireless Communications and Signal Processing (WCSP)*, pp. 1–6, IEEE, 2018.

[50] Y. Dai, D. Xu, S. Maharjan, and Y. Zhang, "Joint load balancing and offloading in vehicular edge computing and networks," *IEEE Internet of Things Journal*, vol. 6, no. 3, pp. 4377–4387, 2018.

[51] D. Tang, X. Zhang, and X. Tao, "Delay-optimal temporal-spatial computation offloading schemes for vehicular edge computing systems," in *2019 IEEE Wireless Communications and Networking Conference (WCNC)*, pp. 1–6, IEEE, 2019.

[52] J. Wang, D. Feng, S. Zhang, J. Tang, and T. Q. Quek, "Computation offloading for mobile edge computing enabled vehicular networks," *IEEE Access*, vol. 7, pp. 62624–62632, 2019.

[53] Y. Bi, "Neighboring vehicle-assisted fast handoff for vehicular fog communications," *Peer-to-Peer Networking and Applications*, vol. 11, no. 4, pp. 738–748, 2018.

[54] H. Matsumoto, B. Gu, and O. Mizuno, "A v2x task offloading method considering automobiles' behavior in urban area," in *2019 20th Asia-Pacific Network Operations and Management Symposium (APNOMS)*, pp. 1–4, IEEE, 2019.

[55] Y. Liu, S. Wang, J. Huang, and F. Yang, "A computation offloading algorithm based on game theory for vehicular edge networks," in *2018 IEEE International Conference on Communications (ICC)*, pp. 1–6, IEEE, 2018.

[56] K. Zhang, Y. Mao, S. Leng, S. Maharjan, and Y. Zhang, “Optimal delay constrained offloading for vehicular edge computing networks,” in *2017 IEEE International Conference on Communications (ICC)*, pp. 1–6, IEEE, 2017.

[57] G. Premsankar, B. Ghaddar, M. Di Francesco, and R. Verago, “Efficient placement of edge computing devices for vehicular applications in smart cities,” in *NOMS 2018 - 2018 IEEE/IFIP Network Operations and Management Symposium*, pp. 1–9, IEEE, apr 2018.

[58] B. Aslam, F. Amjad, and C. C. Zou, “Optimal roadside units placement in urban areas for vehicular networks,” in *2012 IEEE Symposium on Computers and Communications (ISCC)*, pp. 000423–000429, IEEE, 2012.

[59] D. Gangadharan, O. Sokolsky, I. Lee, B. Kim, C.-W. Lin, and S. Shiraishi, “Bandwidth optimal data/service delivery for connected vehicles via edges,” in *2018 IEEE 11th International Conference on Cloud Computing (CLOUD)*, pp. 106–113, IEEE, 2018.

[60] S. E. G. S. of Computer Science and M. K. University, “Guidelines for performing systematic literature reviews in software engineering,” 2007. Available at https://ees.elsevier.com/infsof/img/525444systematicreviewsguide.pdf, last accessed on the 26th of January, 2020.

[61] J. F. Nash *et al.*, “Equilibrium points in n-person games,” *Proceedings of the national academy of sciences*, vol. 36, no. 1, pp. 48–49, 1950.

[62] J. D. Ullman, “Np-complete scheduling problems,” *Journal of Computer and System sciences*, vol. 10, no. 3, pp. 384–393, 1975.

[63] W. Mathworld, “Np-hard problem definition.” Available at http://mathworld.wolfram.com/NP-HardProblem.html, last accessed on the 30th of January, 2020.

[64] J. Harri, F. Filali, and C. Bonnet, “Mobility models for vehicular ad hoc networks: a survey and taxonomy,” *IEEE Communications Surveys & Tutorials*, vol. 11, no. 4, pp. 19–41, 2009.

[65] M. T. Spaan, “Partially observable markov decision processes,” in *Reinforcement Learning*, pp. 387–414, Springer, 2012.

[66] Investopedia, “Nash equilibrium definition,” 2019. Available at https://www.investopedia.com/terms/n/nash-equilibrium.asp, last accessed on the 30th of January, 2020.

[67] J. Kennedy and R. Eberhart, “Particle swarm optimization,” in *Proceedings of ICNN’95-International Conference on Neural Networks*, vol. 4, pp. 1942–1948, IEEE, 1995.

[68] L. Lovász and M. D. Plummer, *Matching theory*, vol. 367. American Mathematical Soc., 2009.

[69] C. Cortes and V. Vapnik, “Support-vector networks,” *Machine learning*, vol. 20, no. 3, pp. 273–297, 1995.

[70] O. Trullols, M. Fiore, C. Casetti, C.-F. Chiasserini, and J. B. Ordinas, “Planning roadside infrastructure for information dissemination in intelligent transportation systems,” *Computer Communications*, vol. 33, no. 4, pp. 432–442, 2010.

[71] I. University, “Understand measures of supercomputer performance and storage system capacity.” Available at https://kb.iu.edu/d/apeq, last accessed on the 8th of June, 2020.

[72] J. Redmon, “Yolo: Real-time object detection.” Available at https://pjreddie.com/darknet/yolo/, last accessed on the 8th of June, 2020.

[73] M. W. Trosset, “What is simulated annealing?,” *Optimization and Engineering*, vol. 2, no. 2, pp. 201–213, 2001.

[74] S. Agarwal, “On ranking and choice models.,” in *IJCAI*, pp. 4050–4053, 2016.

[75] F. Wauthier, M. Jordan, and N. Jojic, “Efficient ranking from pairwise comparisons,” in *International Conference on Machine Learning*, pp. 109–117, 2013.

[76] S. Negahban, S. Oh, and D. Shah, “Rank centrality: Ranking from pairwise comparisons,” *Operations Research*, vol. 65, no. 1, pp. 266–287, 2017.

[77] B. Boehm and W. J. Hansen, “Spiral development: Experience, principles, and refinements,” tech. rep., CARNEGIE-MELLON UNIV PITTSBURGH PA SOFTWARE ENGINEERING INST, 2000.

[78] OpenSim, “What is omnet++?,” 2019. Available at https://omnetpp.org/intro/, last accessed on the 5th of February, 2020.

[79] M. Behrisch, L. Bieker, J. Erdmann, and D. Krajzewicz, “Sumo–simulation of urban mobility: an overview,” in *Proceedings of SIMUL 2011, The Third International Conference on Advances in System Simulation*, ThinkMind, 2011.

[80] OpenStreetMapFoundation, “Open street map.” Available at https://www.openstreetmap.org/#map=7/39.602/-7.839, last accessed on the 12th of June, 2020.

[81] Python.org, “Python website.” Available at https://www.python.org/, last accessed on the 28th of June, 2020.

[82] cplusplus.com, “std::map.” Available at http://www.cplusplus.com/reference/map/map/, last accessed on the 24th of June, 2020.

[83] J. Song, Y. Wu, Z. Xu, and X. Lin, “Research on car-following model based on sumo,” in *The 7th IEEE/International Conference on Advanced Infocomm Technology*, pp. 47–55, IEEE, 2014.

[84] D. Jiang and L. Delgrossi, “Ieee 802.11 p: Towards an international standard for wireless access in vehicular environments,” in *VTC Spring 2008-IEEE Vehicular Technology Conference*, pp. 2036–2040, IEEE, 2008.

[85] S. Media, “What is h.264?.” Available at https://www.streamingmedia.com/Articles/ReadArticle.aspx?ArticleID=74735#:~:text=H.,Video%20Coding%2C%20or%20AVC)., last accessed on the 8th of June, 2020.

[86] F. Tech, “Tcp/ip protocol.” Available at https://cs.fit.edu/~mmahoney/cse4232/tcpip.html, last accessed on the 8th of June, 2020.

[87] F. O. Solutions, "How fast fiber optic cable speed is." Available at http://www.fiber-optic-solutions.com/fast-fiber-optic-cable-speed.html, last accessed on the 8th of June, 2020.

[88] Y. Wang, X. Duan, D. Tian, G. Lu, and H. Yu, "Throughput and delay limits of 802.11 p and its influence on highway capacity," *Procedia-Social and Behavioral Sciences*, vol. 96, pp. 2096–2104, 2013.

[89] Cisco, "Cisco catalyst 9200 series switches data sheet." Available at https://www.cisco.com/c/en/us/products/collateral/switches/catalyst-9200-series-switches/nb-06-cat9200-ser-data-sheet-cte-en.html, last accessed on the 8th of June, 2020.

[90] T. PowerUp, "Nvidia titan x pascal." Available at https://www.techpowerup.com/gpu-specs/titan-x-pascal.c2863, last accessed on the 8th of June, 2020.

[91] M. M. Cruz-Cunha, *Handbook of Research on Mobility and Computing: Evolving Technologies and Ubiquitous Impacts: Evolving Technologies and Ubiquitous Impacts*, vol. 1. IGI Global, 2011.

[92]

[93] R. 0rg, "Raspberry pi 4." Available at https://www.raspberrypi.org/products/raspberry-pi-4-model-b/, last accessed on the 9th of June, 2020.

[94] M. University, "The gflops/w of the various machines in the vmw research group." Available at http://web.eece.maine.edu/~vweaver/group/green_machines.html, last accessed on the 9th of June, 2020.

[95] S. A. Jarvis, S. A. Wright, and S. D. Hammond, *High performance computing systems. Performance modeling, benchmarking and simulation: 4th International Workshop, PMBS 2013, Denver, CO, USA, November 18, 2013. Revised Selected Papers*, vol. 8551. Springer, 2014.

[96] NVIDIA, "Nvidia's next-gen pascal gpu architecture to provide 10x speedup for deep learning apps." Available at https://blogs.nvidia.com/blog/2015/03/17/pascal/, last accessed on the 9th of June, 2020.