Development and Performance Evaluation of Network Function Virtualization Services in 5G Multi-Access Edge Computing


Doctoral Dissertation
Doctoral Program in Electrical, Electronics and Communications Engineering
(33^{rd} cycle)

Development and Performance Evaluation of Network Function Virtualization Services in 5G Multi-Access Edge Computing

Giuseppe Avino
* * * * *

Supervisors
Prof. Carla Fabiana Chiasserini, Supervisor
Prof. Claudio Ettore Casetti, Co-supervisor

Doctoral Examination Committee:
Prof. Antonio Cianfrani, Università degli Studi di Roma "La Sapienza", Rome
Prof. Antonio De La Oliva Delgado, Universidad Carlos III de Madrid, Madrid
Prof. Marcelo Dias de Amorim, Sorbonne Université, Paris
Prof. Fabio Dovis, Politecnico di Torino, Turin
Prof. Renato Lo Cigno, Università degli Studi di Brescia, Brescia

Politecnico di Torino
March 2021

I hereby declare that, the contents and organization of this dissertation constitute my own original work and does not compromise in any way the rights of third parties, including those relating to the security of personal data.

. . . . . .

Giuseppe Avino
Turin, March 2021

Summary

5G technology aims at enriching the telecommunication network ecosystem by improving the existing mobile networks and by supporting the deployment and provision of services from several vertical industries. It is now commonly agreed that three technologies are crucial for the success of 5G, i.e., Network Function Virtualization (NFV), network slicing and Multi-access Edge Computing (MEC). Leveraging these technologies, the 5G-Transformer (5GT) project has developed an open and flexible 5G NFV/MEC-based transport and computing platform tailored to support the tight and heterogeneous requirements of a wide range of vertical services.

In order to assess the effectiveness and reliability of the 5GT architecture, five vertical domains have been selected to develop different use cases (UCs). In this thesis, we focus on the automotive UC, in particular on the design and implementation of a collision avoidance (CA) service (i.e., a road safety service) which provides support to vehicles approaching urban crossroads. The service is based on the periodic and anonymous exchange of messages between vehicles and a CA algorithm hosted in the cellular network infrastructure. By combining the information contained in these messages, the algorithm can detect possible dangerous situations in advance and send unicast warnings to the involved vehicles.

The core of our safety service is indeed the CA algorithm. We have designed a trajectory-based algorithm able to detect collisions between any type of entity, i.e., not only between vehicles, but also between cars and vulnerable road users (e.g., pedestrians). Leveraging this algorithm, we have built a first testbed of a CA service on an *OpenAirInterface* architecture (an open-source software-based cellular network implementation) including MEC functionalities. The service is composed of two main VNFs running in a virtualized environment on the MEC platform: the Cooperative Information Manager (CIM) VNF and the CA VNF. The CIM is a database which decodes and stores the messages sent by vehicles, while the CA VNF runs the trajectory-based algorithm, by relying on the messages stored in the CIM, and generates the warning messages for the cars involved in potentially dangerous situations. We have then assessed the effectiveness and reliability of our CA service through a hardware-in-the-loop simulation technique. We have obtained excellent results, as all the simulated collisions were timely detected (i.e., the alert messages were received by the drivers *sufficiently* in time to react before the collision), with a low number of false positives (i.e., alerts

referring to low or even no danger situations). We have hence shown how the latency and reliability required by delay sensitive safety applications can be addressed by the MEC paradigm and the cellular network.

Finally, once a reliable CA service was developed, we have assessed the performances of the 5GT architecture. In particular, we have tested the automated deployment of our CA application and two important management functions that are provided at the service runtime, i.e., arbitration and scaling. These two functionalities enable the fulfillment of the requirements during the whole lifecycle of services. The architecture developed in the 5GT project has proven to be suitable to meet the significantly different requirements of vertical services, offering a platform on which they can be easily deployed, guaranteeing, at the same time, their service level agreements.

Acknowledgements

Before presenting the work, I would like to thank everyone helped me during this Ph. D. program. First, I would express my profound gratitude to my advisor, prof. Carla Fabiana Chiasserini, and co-advisor, prof. Claudio Ettore Casetti, for their continuous support along these three years. I thank them for their endless scientific knowledge, for their incredible patience, and for every suggestion and teaching. Thanks for believing in me and allowing me to grow professionally.

A huge thanks to my mom, Laura, and my dad, Angelo. Thanks for your enormous effort and sacrifice to allow me to achieve this goal. You have been an inspiration.

I would like to thank all my awesome colleagues of this journey, Greta, Marco, Francesco, Corrado and Kaldidan. Thanks for our coffee breaks and for the shared meals together, thanks for making less painful certain moments on this journey. Especially thanks to Greta and Francesco, your suggestions were of great importance to me in writing this manuscript. Thanks also to Christian and Francesco, two former post-doctoral fellows of Politecnico di Torino, from whom I learned a great deal about scientific research. Thanks also to all the people met in my crazy experience at IMDEA during a pandemic. Special thank to prof. Joerg Widmer, who hosted me for 4 months and still continues to support part of my work. Thanks also to the CRF team with which I worked within the 5G-Transformer project, so thanks to Paolo, Giuliana and Marina.

I would like to thanks my second family, all the friends I grew up with, Dede, Blepty, Nik, Mari, Michi, Agger, Picca and Anes. But also thanks to the "newer" friends, Greta, Vetrus, Lucia and Francesca. Really, you have been fundamental to me. Thanks for each night spent together ("da Gianca"), each beer, each dinner, and our awesome summer vacations. Thanks to my new *bro* Robi and to her husband and my lifelong friend, Bordo. Thanks because you have always been present and ready to help me at any time during these three years.

Thanks to Daniela, Claudio, Silvia and Carmelo, four very important people in my life, people I can always rely on.

Thanks also to all those people who I have not explicitly mentioned but who have contributed, even in a small part, to this work and to my professional training.

Finally, thanks to Michela. You are saving me from the darkest period of my life. Thanks for putting up with me even when no one else would, thanks for your enormous support, for how much you believe in me and for your incredible and awesome *disegnini*.

Dedicato a Debby

alla persona che più di tutte ha creduto in me e che mi ha insegnato che ogni giorno è un Buon Giorno.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

In the last decades, the wireless communications industry has experienced an overwhelming growth. Legacy 3G and 4G/LTE have enabled the expansion of mobile Internet, opening the path to a wide variety of multimedia applications, such as mobile video streaming and gaming, smart cities, and several others. This wide range of new services led to an escalation in the number of users, thus in the total amount of mobile data traffic. According to a study by the International Telecommunication Union (ITU), the global mobile data traffic will increase from the current 0.057 zettabyte (ZB) per month to 5 ZB, in 2030 [97]. The great popularity of mobile Internet and the continuous rise of massively data-intensive UCs and applications, as mentioned in the ITU study, brought to the development of the fifth generation of mobile networks (5G), and they will be the driving force for the future ones.

With the development of the 5G technologies, an increasing number of vertical industries such as automotive, robotics, eHealth, is interested in developing state-of-the-art services, which demand stringent requirements in terms of latency, reliability, data rate, coverage, power consumption. In order to meet these requirements and the massive device connectivity, 5G systems must be efficient in terms of energy consumption and resource management, and they should provide a high scalability and versatility, with the latter constituting one of the main challenges undertaken by industry and research when developing the technologies beneath 5G. These challenges can be tackled by implementing the 5G network functions (NFs) as software components, by relying the Network Function Virtualization (NFV) paradigm [68]. In this context, hardware-based network functions become software-based, implemented as virtual network functions (VNFs) in general purpose *telco-cloud* instead of specialized hardware. The virtualization of NFs makes it possible to cope with the continuously increasing traffic demands because it enables the creation of elastic on-demand networks along with their lifecycle management, ensuring the scalability and versatility required by the 5G networks.

Along with NFV, it is fundamental to mention two other pillar technologies in 5G

networks: Multi-access Edge Computing (MEC) [54] and network slicing [9]. MEC provides IT services and cloud-computing capabilities at the edge of the mobile network, within the Radio Access Network (RAN). Network slicing, on the other hand, enables the subdivision of the infrastructure in slices, each one hosting a particular service. Each network slice can then be configured in order to meet the specific requirements of the service will be deployed there. Given its flexibility, the 5G community considers network slicing one of the most valid and cost-efficient solutions to share the mobile network infrastructure among 5G services.

NFV, MEC and network slicing are deemed pivotal technologies not only for 5G but also for the next generations of mobile networks. The future sixth generation will be enhanced by many new driving features, but there is a broad consensus among researchers that it will continue to benefit from many 5G technologies.

1.1 Key enablers for 5G and next generation mobile networks

5G is envisioned to expand and improve the existing mobile networks by optimizing the throughput, latency, energy consumption, reliability and coverage. The high performances of the mobile networks allow vertical industries to deploy a wide range of services, including the ones with stringent requirements (e.g., delay sensitive services). As described above, there is a broad consensus in considering some technologies crucial for the accomplishment of 5G and next generation mobile networks. In this section, a brief overview of the main key enablers is presented.

1.1.1 Network virtualization

One of the possible approaches to adapt the network architecture to the multi-provider nature of the Internet and to its high number of users is *network virtualization*. The multiplicity of 5G services and their extremely different performance requirements cannot be supported in an efficient way by relying on the current architecture. A candidate technology, based on network virtualization and emerged to address this challenge, is *network slicing*. Network slicing envisions to divide the infrastructure in virtual logical subnetworks called *network slices*. Each network slice represents an isolated and independent virtualized end-to-end network, tailored to meet the performance requirements of the supplied services.

In this new vision, services are deployed on network slices as set of individual softwarized network functions (i.e., VNFs). The logical connectivity of VNFs belonging to the same service, as well as their allocation onto the infrastructure, is described by the so-called VNF forwarding graph (VNFFG). The decomposition of services in function blocks enables the sharing of virtualized functions among different slices, and it more generally ensures:

- **Optimization of the resource allocation**, both in terms of energy and costs;
- **Migration** of VNFs from one hardware to another;
- **Scalability** of VNFs to let the service adapt to network conditions;
- **High reliability in performance guarantees of VNFs operations**, including maximum latency and packet loss;
- **Coexistence** of VNFs with non-virtualized functions.

This new and flexible virtual architecture turns out to be essential in modern mobile networks, strongly characterized by the provision of heterogeneous services with diverse stringent requirements. For this reason, despite the numerous challenges posed, network slicing is considered a pillar for modern mobile networks.

1.1.2 NFV management and orchestration framework

The NFV paradigm enables the decomposition between the implementation of network functions and the underlying hardware. This concept is, instead, completely different from the approach used in legacy networks, where the implementations of NFs is tightly coupled with the infrastructure. According to the European Telecommunications Standards Institute (ETSI), NFV adds new capabilities to communications networks and requires a novel set of management and orchestration functions to be added to the current model of operations, administration, maintenance and provisioning [34]. These new functions aim to coordinate the NFs and the infrastructure they run on. The solution proposed by ETSI is the so-called NFV Management and Orchestration (NFV-MANO) framework, an entity in charge of managing the infrastructure and allocating the resources needed by the VNFs to run each service.

The MANO platform represents the key enabler for the service and infrastructure virtualization, as it manages the instantiation and termination of VNFs by, respectively, allocating and releasing the physical resources. Besides the resource allocation, it is also responsible for all the operations that are part of the VNFs' lifecycle, such as the VNF *scale* and VNF *update*. The former allows a VNF to increase or reduce their computation and storage capabilities at runtime, while the latter refers to the possibility of changing the VNF configuration.

As the reader can realize, the concept of service manager and orchestrator is central to NFV. In the last years, numerous orchestration platforms have been proposed, such as OpenSource MANO (OSM) [77], Cloudify [29] and Open Network Automation Platform (ONAP). In Section 1.2.2 the 5G-Transformer service orchestrator platform is presented. This orchestrator offers additional features with respect the previously

mentioned platforms, including the support for network slicing, MEC deployment and service federation[1].

1.1.3 Multi-access Edge Computing

MEC paradigm envisions the instantiation and execution of services at the edge of the network, in close proximity to mobile subscribers. The main features of a MEC environment are:

- **Low latency**. Ultra-low delays are ensured by the service location, within the RAN, very close to the end users.
- **Reduction of core traffic**. Running applications at the RAN level means handling traffic at the network edge, thus reducing the load at the core network.
- **Network information**. MEC applications can exploit real-time radio and network information to offer context-related services.
- **Location information** Users' positions and mobility patterns can be collected by base stations at the edge of the network and leveraged on by MEC applications.
- **Proximity**. MEC servers are physically close to the final users. Their proximity to user equipments (UEs) make them particularly proficient in gathering analytics and information for big data.

Generally, such features enable an increased quality of experience of mobile subscribers and ensure efficient network and service operations based on real-time radio, network, and location information.

Multi-access Edge Computing needs the support of various key technologies. The deployment of high volume servers at each mobile station is an absolute necessity to develop a MEC architecture, as well as *virtualization*. Indeed, each MEC application is virtualized and runs on top of a virtualization infrastructure located at the network edge. The virtualization infrastructure, along with the MEC platform and MEC applications, is part of the MEC host, the core of the MEC paradigm. Further details about the MEC architecture, its components and interfaces will be provided in Chapter 4.

Multi-access Edge computing moves computation and storage capacity from the core to the points of access of the network. In spite of various challenges that MEC deployments pose, such as resource management, network integration, security and privacy, the offered technological opportunities make this paradigm essential for the present and future mobile networks ecosystem.

[1]Federation: the possibility to deploy a network service in the administrative domain owned by another operator.

1.2 The 5G-Transformer architecture

Most of the research activity described in this manuscript is related to the European Project 5G-Transformer (5GT) [4]. Politecnico di Torino was involved in this project along with many relevant European universities, research centers and companies. The aim of the 5GT project is to deploy an open and flexible 5G transport and computing platform tailored to support diverse service requirements of various vertical industries [66]. The architecture is based on the ETSI MANO platform and presents innovative functionalities. The purpose of the architecture is twofold: firstly, to enable the deployment of vertical services within network slices meeting their performance requirements and, secondly, to manage network slices throughout a federated virtualized infrastructure.

In the context of the 5GT project, a few vertical industries, i.e., automotive, eHealth, media and entertainment, and e-Industry, designed and implemented services with different performance requirements. When instantiating a vertical service, the 5GT platform creates an end-to-end 5GT slice, i.e., a dedicated logical infrastructure on which vertical services are deployed meeting the specific requirements of the customer. 5GT slices are therefore composed of a set of VNFs and/or virtual applications (VAs) with the resources needed to the service. Figure 1.1 shows the 5GT concept and highlights the three main components of the architecture:

- **Vertical Slicer** (5GT-VS),
- **Service Orchestrator** (5GT-SO),
- **Mobile Transport Platform** (5GT-MTP).

The 5GT architecture is described more in details in the following subsections.

1.2.1 The 5GT Vertical Slicer

The 5GT-VS is a new functional block introduced in the 5GT architecture, representing the common entry point for verticals. It is part of the operating and business support system (OSS/BSS) of the administrative domain of a provider.

Vertical users interact with the VS whenever they want to deploy a new service. To achieve this goal, verticals should prepare specific templates or *blueprints* containing high-level details of the required service. The blueprints are designed in a way which is easily understandable by verticals, and they may be integrated with essential services provided by the platform. The resulting descriptor is called vertical service descriptor (VSD) and is mapped by the VS onto a network service descriptor (NSD). The NSD is hence a service graph composed of VNFs chains and other crucial instantiation parameters (e.g., deployment flavor) used by the SO to instantiate the service.

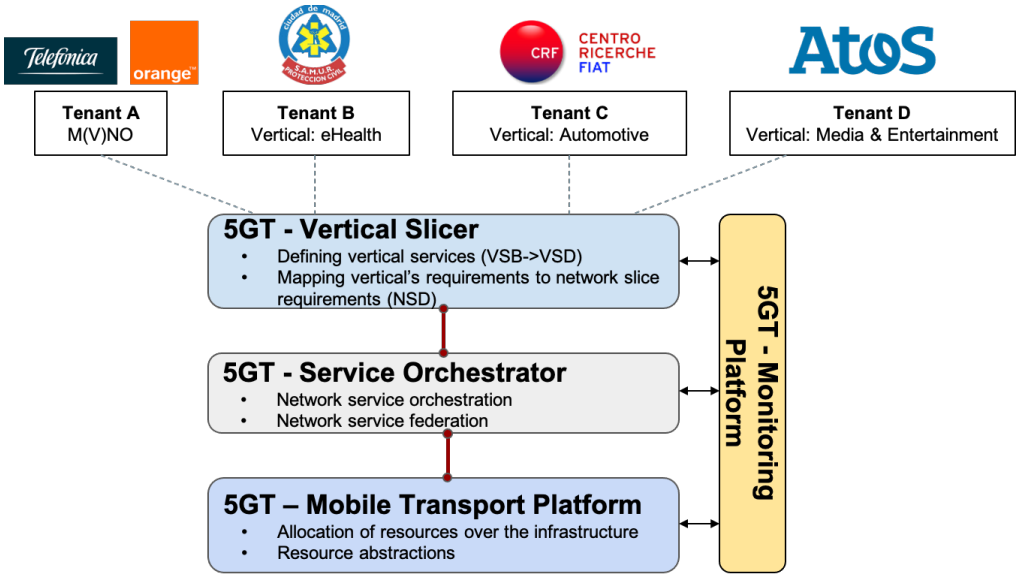


Figure 1.1: 5G-Transformer concept.

Another important function performed by the 5GT-VS is service *arbitration*. An arbitration mechanism enables the handling of services of a certain vertical according to their service level agreement (SLA) requirements, service priorities and available resources.

The 5GT-VS enables vertical industries to easily deploy services such as VNFs chains, thanks to the blueprints mentioned earlier, which, being simple interconnection models, hide all the low-level slice details from the customer. Being the complexity hidden at the VS level, verticals can deploy services in short time-scale with an easy-to-use mechanism.

1.2.2 The 5GT Service Orchestrator

The 5GT-SO is the core of the system, being the entity that provides end-to-end orchestration of services across multiple administrative domains. The need of a management and orchestration platform in NFV was widely introduced in the previous section. Here, we describe in detail how the 5GT orchestrator works and which functionalities it can provide.

The 5GT-SO interacts with the 5GT-MTP and with the service orchestrators of the other administrative domains. It manages the allocation of virtual resources to network slices. Upon the reception of a request from the VS, the SO maps the received NSD to an MTP network slice and it decides whether to create a new slice or to rely on an existing one, by leveraging on resource sharing. Requested can come both from the 5GT-VS and from the mobile (virtual) network operator (M(V)NO).

When instantiating a service, if the 5GT-SO detects that the MTP does not have enough resources to ensure the desired SLA, it contacts other SOs to compose service

federation. In this case, the SO will interact with the neighboring SOs and the service will be orchestrated over different administrative domains. Federation is completely hidden to verticals.

Finally, the 5GT-SO also implements a flexible monitoring platform (5GT-MON) to efficiently check whether the service requirements are met. The 5GT-MON works on multiple domains to collect data from various MTPs and to support service management at runtime. The monitoring data acquired by the platform is available to the verticals through a set of dedicated APIs.

1.2.3 The 5GT Mobile Transport Platform

The 5GT-MTP is the actual infrastructure over which VNFs and physical network functions (PNFs) are deployed. The MTP orchestrates the resources, handles the instantiation of VNFs, and manages the physical mobile transport network and the computing and storage infrastructure. Furthermore, it provides a MEC platform for the deployment of sensitive low-latency services and support for network slicing. The MTP interacts with the SO and exposes to it an abstract view of the available computation and storage resources. Thanks to this feature, the SO can select the proper resources to allocate vertical slices. As mentioned above, if the resources available on the MTP are not enough, the service is orchestrated in federation across multiple administrative domains.

Overall, the 5GT-MTP aims at providing a complete and scalable MTP integrated with MEC services and supporting the dynamic placement and migration of VNFs. Additionally, the platform introduces innovative mechanisms to effectively share VNFs among different services.

1.3 Road safety applications

As previously mentioned, the 5GT consortium includes verticals from different industries, such as automotive and eHealth. In order to accomplish the strict constraints and satisfy the high demand of their applications, each one of these industry partners is undergoing key technological transformations and changes, effectively taking advantage of 5G networks. As a result, they were deemed suitable for the purposes of the project and selected for demonstration.

Within this project, the efforts of Politecnico di Torino mainly focused on the automotive use case (UC), in particular on the design and development of a road safety application. The focus of innovation of the automotive industries is shifting towards the connected and fully automated (i.e., autonomous) vehicle. A vehicle able to communicate with the surrounding environment (including roadside infrastructure elements, vulnerable road users, other vehicles) can help the driver or the vehicle itself making more informed decisions, based on exchanged local views and information from nearby entities, instead of rely only on local awareness based on on-board sensors.

As a result, to be *connected* is an essential requirement for an autonomous vehicle and to drastically reduce fatalities on the road and improve traffic flow. To enable such an idea, many road safety applications have been designed, such as cooperative sensing, collision avoidance, or high-density platooning. These types of safety applications feature hard-to-meet communication requirements, well beyond the legacy 4G/LTE networks or IEEE 802.11 standard. An ultra-low latency below 10 ms, ultra-high reliability close 100% and a high-data rate in the order of Gbps, are the main features indicating the need for 5G networks [71].

In this manuscript we focus and present the automotive UC within the 5GT project, i.e., a vehicle collision avoidance application which offers support to vehicles approaching blind crossroads.

1.4 Main contributions

The main purpose of this thesis is the design, development and performance evaluation of a road safety service suitable for the 5GT architecture. Firstly, we investigated on container-based virtualization technologies since services instantiated on the 5GT platform are composed of VNFs running as software components. Then, we designed, implemented and assessed the performances of the road safety service, both in simulation and within the 5GT architecture. Finally, we validated a couple of key functionalities of the 5GT platform for the management of vertical services (focusing on the automotive domain). Further details about the main contributions and the topics covered in this work are presented below.

Suitability of the Docker framework for a 5G architecture

In the 5GT architecture, both vertical and network services can be instantiated as virtual functions (VFs). Isolation, service scalability and live migration are just a few of the main advantages brought by virtualization. The main existing virtualization technologies are either hypervisor-based or container-based. Both approaches require additional resources to be available, and their overheads may negatively impact the resource utilization as well as the quality of service.

For many years virtual machines (VMs) have been the most widespread virtualization technology. Recently, container-based solutions have become appealing as a valid alternative to VMs. This great and raising popularity mainly comes from the Docker framework [31]. This platform extended the Linux container technology in various ways (for instance, by introducing a user-friendly interface), providing the users with a complete solution for the management of containers' lifecycle. Docker claims to be a lightweight containerization solution and a perfect tool for enabling an easy and fast deployment of applications. This led us to carry on a study on the suitability of Docker in a 5G architecture by quantifying the CPU consumed by Docker when running two different containerized services: multiplayer gaming and video streaming. We selected

these two applications because they clearly represent service models with opposite requirements in terms of CPU load and managed data. Throughout our experiments, we validated the Docker framework as a lightweight virtualization tool and we studied the overhead resource consumption trend when varying both the number of users consuming the services and the number of virtualized servers.

Low overheads are pivotal in the 5G ecosystem where plenty of services are virtualized. In this context, Docker exhibited excellent performances, proving to be a valid virtualization alternative to VMs. However, due to some challenges and issues which arose in the interaction between Docker containers and the 5GT architecture, we preferred to opt for the hypervisor-based solution, virtualizing network and vertical services with VMs.

Within the container virtualization area, our contributions can be found in:

- *Giuseppe Avino, Marco Malinverno, Francesco Malandrino, Claudio Casetti, Carla Fabiana Chiasserini. "Characterizing Docker Overhead in Mobile Edge Computing Scenarios".* Published in *Proceedings of the Workshop on Hot Topics in Container Networking and Networked Systems. p. 30-35, 21-25 August 2017, Los Angeles, CA, USA.* https://doi.org/10.1145/3094405.3094411

- *Francesco Malandrino, Carla Fabiana Chiasserini, Giuseppe Avino, Marco Malinverno, Scott Kirkpatrick. "From Megabits to CPU Ticks: Enriching a Demand Trace in the Age of MEC".* Published in *IEEE Transactions on Big Data (2018). https://ieeexplore.ieee.org/document/8447497*

Design and evaluation of a collision avoidance algorithm

5G networks aim to improve the existing mobile networks and support several vertical services, including the ones with strict performance requirements. The automotive industry is one of the verticals acting as a driving force to construct this new ecosystem. With 5G networks, automotive industries will be able to design and deploy road safety applications, which requirements (i.e., ultra-high reliability and ultra-low latency) are not ensured by legacy 4G/LTE networks.

The automotive industry is also one of the main actors in the 5GT system. The automotive service use case in the 5GT project is a *collision avoidance* (CA) service, which provides support to vehicles approaching urban crossroads, especially in non-line-of-sight (NLOS) conditions, which are relatively common in urban areas. The service is based on the periodic [92] and anonymous [63] transmission of *Cooperative Awareness Messages* (CAMs) by vehicles. Each CAM contains information of the sender, including its position, direction, speed and acceleration. CAMs are sent to the CA service, which combines the information contained in the messages from different vehicles and determines if any couple of cars is on a collision course.

The first step to implement the CA service is the development of an efficient collision avoidance algorithm. We designed a trajectory-based algorithm that can be applied to

any kind of possibly colliding entity (i.e., not only to vehicle-with-vehicle collisions but, for instance, also to vehicle-with-pedestrian collisions). The algorithm takes as input position, direction, speed and acceleration of two road users and determines *(i)* the time instant at which the distance between them will be the minimum one and *(ii)* such a minimum distance. If these values are lower than a threshold, the system generates and forwards a warning message to the two vehicles. The algorithm was extensively tested in a simulation environment, by relying on the SimuLTE-Veins simulator [90]. The simulation-based results highlighted the reliability of our algorithm, which was able to detect in advance all the simulated collisions, and, more importantly, to generate *sufficiently* on time the warning messages, leaving to the drivers a large enough time margin to react.

Part of the CA algorithm development was carried out with *TIM*, within a research project for the study of MEC-based services for road users.

In this field, our research activity can be found in:

- *Giuseppe Avino, Marco Malinverno, Francesco Malandrino, Claudio Casetti, Carla Fabiana Chiasserini, Giovanni Nardini, Salvatore Scarpina. "Poster: A Simulation-based Testbed for Vehicular Collision Detection".* Published in *the IEEE Vehicular Networking Conference (VNC), 27-29 November 2017, Turin, Italy.* https://ieeexplore.ieee.org/document/8275655

- *Marco Malinverno, Giuseppe Avino, Claudio Casetti, Carla Fabiana Chiasserini, Francesco Malandrino, Salvatore Scarpina . "Performance Analysis of C-V2I-based Automotive Collision Avoidance".* Published in *the 19th IEEE International Symposium on "A World of Wireless, Mobile and Multimedia Networks" (WoWMoM 2018), 12-15 June 2018, Chania, Greece.* https://ieeexplore.ieee.org/abstract/document/8449772

- *Giuseppe Avino, Marco Malinverno, Claudio Casetti, Carla Fabiana Chiasserini, Francesco Malandrino, Marco Rapelli, Giuliana Zennaro. "Support of Safety Services through Vehicular Communications: The Intersection Collision Avoidance Use Case".* Published in *the IEEE International Conference of Electrical and Electronic Technologies for Automotive, p. 1-6, 9-11 July 2018, Milan, Italy.* https://ieeexplore.ieee.org/abstract/document/8493191

- *Marco Malinverno, Giuseppe Avino, Claudio Casetti, Carla Fabiana Chiasserini, Francesco Malandrino, Salvatore Scarpina. "MEC-based Collision Avoidance for Vehicles and Vulnerable Users".* Published in *IEEE Vehicular Technology Magazine, vol. 15, no. 1, pp. 27-35, March 2020.* https://doi.org/10.1109/MVT.2019.2953770

Design, implementation, and performance evaluation of a road safety service through a testbed implementation

Simulations were only the first step for the design of the 5G-Transformer road safety service. Once the reliability of the CA algorithm had been verified through simulations, we could start the design and implementation of a full-fledged testbed. Our testbed has been built by leveraging on the popular OpenAirInterface (OAI) project [78] (an open source implementation of a full mobile network), and by implementing a flexible MEC architecture to support of ultra-low latency services.

The CA application, which was designed together with *Centro Ricerche Fiat*, a research branch of *Fiat Chrysler Automobiles* (CRF-FCA), is a service composed of two main VNFs: the Cooperative Information Manager (CIM) and the CA algorithm. The two VNFs run on two different VMs on top of the MEC platform. The CIM is a database in charge of receiving and storing the CAM messages coming from the vehicles. The CA algorithm, instead, consumes the information contained in the CAMs to detect in advance collisions between vehicles (it leverages on the algorithm presented earlier). All the vehicles are simulated by a software, which generates and transmits the same CAMs that real vehicles would send. The service instantiation on our MEC platform (which represents the 5GT-MTP) is handled by the 5GT architecture. In order to make this operation possible, we prepared a blueprint template for the CA service, which, as described in Section 1.2, is used by the 5GT-VS during the service on-boarding.

The 5GT architecture correctly instantiated the CA service on a 5GT slice of the MEC platform and the experimental measurements obtained through our testbed revealed the excellent performance of our MEC-based road safety service. We evaluated the reliability of the application by considering the number of collisions correctly and timely detected and the number of generated *false positives*, i.e., warning messages referring to low or even no danger. The service was able to detect every collision occurred in the experiments and, at the same time, to generate a low number of false positives. This second aspect is also fundamental to establish user confidence in the reliability of alerts received through the system.

Finally, to prove the reliability of the CA service also in real-word scenarios, we made several field tests with actual cars. The trials were conducted in a test circuit with two vehicles and expert drivers. The two cars were equipped with Uu interfaces for the communication with our testbed, and with an automatic braking system. The two drivers were instructed to simultaneously approach a crossroad, creating a collision risk situation. The CA service, combining the information received in the CAMs transmitted by the vehicles, could detect in advance all the actual collisions and generate the corresponding warning messages. The outcome of these test sessions was excellent, as all the collisions were avoided thanks to the timely reception of the warning messages, which properly triggered the automatic braking system.

The testbed and service implementation, as well as the service performance evaluation, were broadly reported in:

- *Giuseppe Avino, Paolo Bande, Pantelis A. Frangoudis, Christian Vitale, Claudio Casetti, Carla Fabiana Chiasserini, Kalkidan Gebru, Adlen Ksentini, Giuliana Zennaro. "A MEC-based Extended Virtual Sensing for Automotive Services".* Published in *IEEE Transactions on Network and Service Management, pp.1450-1463, July 2019.* https://ieeexplore.ieee.org/abstract/document/8781832

Techniques for effective instantiation and adaptation of vertical services: design and implementation of service *arbitration* and *scaling*

After having designed, implemented and assessed the performance of an automotive service, we focused on the validation of some key 5GT platform functionalities.

In general, the 5GT architecture facilitates the instantiation of vertical services, manages the lifecycle of the VFs composing them, and guarantees the fulfillment of the service requirements. As far as this last aspect is concerned, two features of the platform are of great importance: the service *arbitration* and the service *scaling*.

As described in Section 1.2.1, the arbitration function is performed by the 5GT-VS component and enables the handling of different service requests coming from a certain vertical while satisfying the agreed SLAs between the service provider (SP) and the vertical itself. We validated this functionality, as offered by the 5GT architecture, by considering two relevant automotive services: a high priority service, i.e., the CA service, and a low priority one, i.e., a video streaming. The validation is divided into three phases. First, the video streaming service is instantiated on the 5GT platform. Then, once the video server is up and running, the request for the instantiation of the CA service is performed and the arbitrator module checks the available resource budget for this vertical. However, it realizes that such budget is not sufficient to run both services in parallel. This situation may occur since the available resources for each vertical are limited and agreed with the infrastructure provider. In the third phase, the arbitrator module decides to terminate the running video streaming service since it has a lower priority with respect the safety service. The termination of the this service frees up part of the resources, which can then be used to instantiate the CA service. This arbitration mechanism allows the 5GT platform to effectively manage services belonging to a same vertical user. As shown in this simple case study, according to the priority of each service, the arbitration module chooses which application can run and which should, instead, be terminated when the amount of resources are not sufficient for running more services in parallel.

Another important functionality of the 5GT architecture, which we investigated, is the service scaling. The scaling function is fundamental in order to guarantee the service quality requirements during the whole service lifecycle, under any network operational conditions. One of the main requirements of the CA service is low latency. Indeed, high delays lead to a lowered efficiency of this application, as the generated warning messages would be received late or even after a collision. For this reason, the CA service is instantiated in the MEC platform, even though this may not be sufficient

to ensure the desired latency requirements. As a matter of fact, the overall service delay is composed by both the network latency and the processing time of the service VNFs. The latter is extremely sensible to the service workload; as the higher is the number of users, the higher is the VNFs' processing time. In order to tackle this challenge, the 5GT platform is able, by monitoring the CPU consumed by a service, to autonomously create new instances of the VNFs composing it (*scale out* operation), as well as terminate them (*scale in* operation) once the network is no more in a peak situation.

In order assess the performance of the service scaling provided by the 5GT architecture, in this work we analyzed together the CPU consumed by the CA service and the number of monitored vehicles.

Arbitration and scaling are pivotal functionalities for 5G and the future generation of mobile networks. They ensure to vertical industries both the provision of the high priority services, as well as the their adaptation to any network condition. Our main contributions in the investigation of these topics are included in:

- *Jorge Baranda, Giuseppe Avino, Josep Mangues-Bafalluy, Luca Vettori, Ricardo Martínez, Carla Fabiana Chiasserini, Claudio Casetti, Paolo Bande, Marina Giordanino, Marco Zanzola. "Automated deployment and scaling of automotive safety services in 5G-Transformer".* Published in *the IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN), p. 1-2, 12-14 November 2019, Dallas, TX, USA.* https://ieeexplore.ieee.org/abstract/document/9039990

- *Jorge Baranda, Josep Mangues-Bafalluy, Luca Vettori, Ricardo Martínez, Giuseppe Avino, Carla Fabiana Chiasserini, Corrado Puligheddu, Claudio Casetti, Juan Brenes, Giada Landi, Koteswararao Kondepu, Francesco Paolucci, Silvia Fichera, Luca Valcarenghi. "Demo Abstract: Arbitrating Network Services in 5G Networks for Automotive Vertical Industry".* Published in *the IEEE Conference on Computer Communications Workshops (INFOCOM WORKSHOPS), p 1318-1319, 6-9 July 2020, Toronto, ON, Canada.* https://ieeexplore.ieee.org/document/9162679

- *Giada Landi, Pietro Giardina, Marco Capitani, Koteswararao Kondepu, Luca Valcarenghi, Giuseppe Avino. "Demo: provisioning and automated scaling of network slices for virtual Content Delivery Networks in 5G infrastructures".* Published in *Proceedings of the Twentieth ACM International Symposium on Mobile Ad Hoc Networking and Computing, p. 397-398, 2-5 July 2019, Catania, Italy.* https://dl.acm.org/doi/abs/10.1145/3323679.3326613

1.5 Outline of the thesis

The rest of the thesis is organized as follows:

Chapter 2 focuses on virtualization. It provides a comparison between the hypervisor-based and the container-based solution, and an overview on the most popular containerization technology, i.e., the Docker framework. The chapter ends with the presentation

of some experimental results.

Chapter 3 introduces the automotive vertical industry and our collision avoidance service. It hence presents the core of the service, i.e., the CA algorithm that we designed, as well as the performance evaluation through simulations. The performance of the service was assessed both when it is deployed in a MEC fashion and in the cloud, and both for the detection of vehicle-with-vehicle and vehicle-with-pedestrian collisions. Moreover, this chapter also includes a study in which the service protection is extended also to vulnerable users.

Chapter 4 firstly presents the implementation of a testbed built on the popular OpenAirInterface framework. Then, it describes how the road safety service is designed and implemented in the testbed, and it ends with the service performance evaluation.

Chapter 5 focuses on the automated deployment of services on the 5GT platform, and on two key network functionalities: the scaling and the arbitration. The chapter continues with a more detailed overview of these functions and concludes by presenting the benefits brought to our safety service.

Chapter 6 aims at summarizing the work and the contributions presented in this manuscript.

Chapter 2

Virtualization of Vertical Services through Docker Containers

The 5GT architecture leverages the network slicing, NFV and MEC paradigms, which are considered the pillar technologies for the present and future mobile networks. In this context, both vertical and network services can be virtualized, running as software components. Since our goal is the design and implementation of a road safety application within the 5GT architecture, our first priority was hence to find the best virtualization solution that met our needs. Currently, the main virtualization technologies are either hypervisor-based, an established and extremely reliable solution, or container-based, a relatively new approach compared to the previous one, promising to noticeable reduce the overhead of virualization.

In this chapter, we first discuss the need of a new mobile network architecture able to meet the different and heterogeneous vertical service requirements, and highlight the differences between the two virtualization approaches. Then we focus on Docker, the most popular platform for the creation and management of containers. Finally, we present our experimental results on the Docker overhead CPU consumption when virtualizing two different services: multiplayer gaming and video streaming.

2.1 Service virtualization

With 5G networks, telecom industry is changing its service delivery model, moving from a "horizontal" model in which services are independent of the consumers, to a "vertical" model where services are tailored to each specific industry. With this new approach, the relationship between MNOs and vertical industries is undergoing a complete revolution. Verticals can now interact with the infrastructure provider to instantiate dynamic services and enjoy the ultra-low latency and high throughput offered by the cellular network. The transition to this novel service delivery model is expected to have a great impact on the whole business of mobile networks. Innovative services,

offered by industries that were not covered by 4G, can now be provided thanks to the support of several new use cases. This enriched telecom ecosystem includes a wide range of verticals which can supply various types of services with very different network requirements. The key performance indicators (KPIs) resulting from the main vertical use cases and applications are listed in Table 2.1.

Table 2.1: KPIs of the main vertical use cases and applications [85].

Metric	Most demanding value	Some relevant verticals
Latency	$\leq$ 5 ms	Media and entertainment
Reliability	99.999%	Manufacturing, health
Density	1M terminals/km2	Energy and utilities
Mobility	500 Km/h	Automotive
Slice deployment time	$\leq$ 90 min	All
Data rate per user	$\geq$ 50 Mb/s	Media and entertainment
Location accuracy	$\leq$ 1 m	Public safety

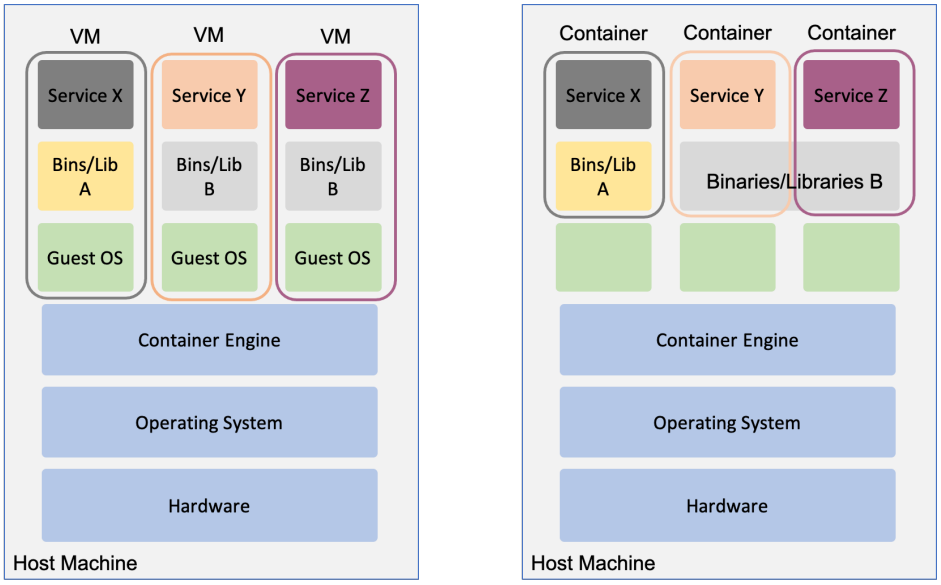
The design and development of a versatile network able to support services with extremely different requirements has thus become a primary need. Network slicing, NFV and MEC, which leverage on the network virtualization paradigm, are considered the major key enablers to meet these diverse service requirements. Network slicing is essential to deploy isolated slices for different services and ensure low latency, high data rates, and high reliability. NFV enables the virtualization of network functions that will be run on network slices, whereas the deployment of services at the edge of the network (MEC paradigm) ensures ultra-low latency to services.

With this aim, the 5GT project has proposed a flexible 5G transport and computing platform tailored to support the vertical services requirements. In the 5GT architecture, which leverages the three technologies mentioned above, both vertical and network services are virtualized, becoming VNFs, and running as software components on network slices. Finding the best solution to virtualize services within the 5GT architecture has been thus one of our priorities. Consequently, in the rest of this section, we provide a brief overview of the two aforementioned virtualization solutions and describe their main differences.

2.1.1 Hypervisor-based vs. container-based virtualization technologies

The *de facto* solution to virtualize environments was, for decades, the hypervisor-based technology. In this virtualization solution, the hypervisor manages the physical resources of the machine and enables the creation of VMs in isolated slices of the hardware. Two types of hypervisors can be mentioned: the *bare-metal hypervisors*, that are

installed and run directly on the computing hardware, and the *hosted hypervisors*, which require a host operating system (OS). Examples of the first type of hypervisors are Xen [110] and VMware's ESX [106], whereas VirtualBox [103] and VMware Server [105] are examples of the second type. To better highlight the differences between containers and VMs, and because of the major popularity of VirtualBox and VMware, in this analysis we will focus on the hosted hypervisors. Such a type of hypervisors provides access to physical resources only, requiring to each VM a full implementation of a guest OS. On the contrary, the container-based solutions virtualize at OS level and share resources with the OS of the physical machine.



(a) Services virtualized in VMs.

(b) Services virtualized in containers.

Figure 2.1: Architectures of hypervisor-based and container-based virtual services [72].

Figure 2.1 shows three applications running in separate VMs and containers. In Figure 2.1a each application runs in a VM, which is created and managed by the hypervisor. The hypervisor also controls the access to the underlying OS and hardware, and interprets system calls. As mentioned above, each VM requires a full copy of the OS. In contrast, Figure 2.1b shows the same three applications virtualized in a containerized system. The container engine is responsible for the creation and termination of the containers, similarly to the hypervisor on VMs. However, unlike VMs, the kernel of the machine is shared with the running containers [72]. This brings two main benefits. First, a much more efficient resource utilization because there is no need to create a whole OS, and applications using same libraries can share this data avoiding redundant copies. Second, containers appear lightweight, therefore fast to create and destroy since there is no need to boot and shutdown an OS. Starting and stopping a container is a matter of few seconds, a much shorter time with respect the one required to do the

same with VMs.

Although several studies have investigated on the performance advantages of containers, such as [107, 109], only with the emergence of Docker the container-based virtualization solutions have gained popularity. Docker is an open source framework that facilitates the management of containers. It has quickly become the most widespread container solution through offering a unified tool set and API for the deployment of Linux containers.

2.2 Docker

Although the interest in containers has particularly grown in recent years, containers are an old concept. In the late 1970s, Unix developed the `chroot` command to provide a basic form of filesystem isolation. In later years, FreeBSD [44] expanded the concept of the chroot command to make it useful for virtualization, whereas, in 2001, SWsoft (now Parallels, Inc.) released the *Virtuozzo container* tecnology for Linux. Then, in 2008, the Linux Containers (LXC) project started and, by leveraging *cgroups*, kernel *namespaces*, and chroot technology, aimed to provide a complete containerization solution [72]. Based on LXC, the first version of Docker was released in 2013. Docker marked a turning point in container virtualization, making it a promising alternative to VMs. It extended the LXC technology by adding several features, such as a user-friendly interface, in order to offer a complete solution to easily create, manage and distribute containers.

Docker is composed of two main components: the Docker engine and the Docker Hub. The former provides a user-friendly interface to create, run and manage containers. It is one of the strengths of Docker, fundamental to its popularity because allows users to work with containers even without specialist knowledge. The Docker Hub is instead a cloud service that provides public container images. This service has been essential for Docker's popularity because allows users to quickly get started and avoid repeating works already done by others.

Following, we describe the Docker architecture with its main components and provide an overview on the building process of Docker images. A rigorous definition of these aspects are out of the scope of this thesis, and further details can be found in theory books like [72].

2.2.1 The Docker architecture

This section addresses the typical Docker architecture. Figure 2.2 shows the key components of a Docker installation:

- **Docker daemon**: It is in charge of creating and running containers, and more in general, managing their lifecycle. The daemon is also responsible for the building and storing of Docker images.

- **Docker client**. It is used by users to interact with the Docker engine. The communication is based on the Hypertext Transfer Protocol (HTTP) and the API used is well documented. Therefore, developers can easily write programs to interact with the daemon and bypass the Docker client.
- **Docker registries**. They are used to store and distribute images. The default Docker registry is the aforementioned Docker Hub, from which users can download "official" images. However, Docker gives also the possibility to build and run private registries.
- **Docker objects**. They are diverse entities used to assemble applications in Docker. The main classes of objects are the following:
  - Docker container: standardized and encapsulated environment where applications run.
  - Docker image: read-only template needed to build containers.
  - Docker service: to enable the scaling of containers across multiple Docker daemons.

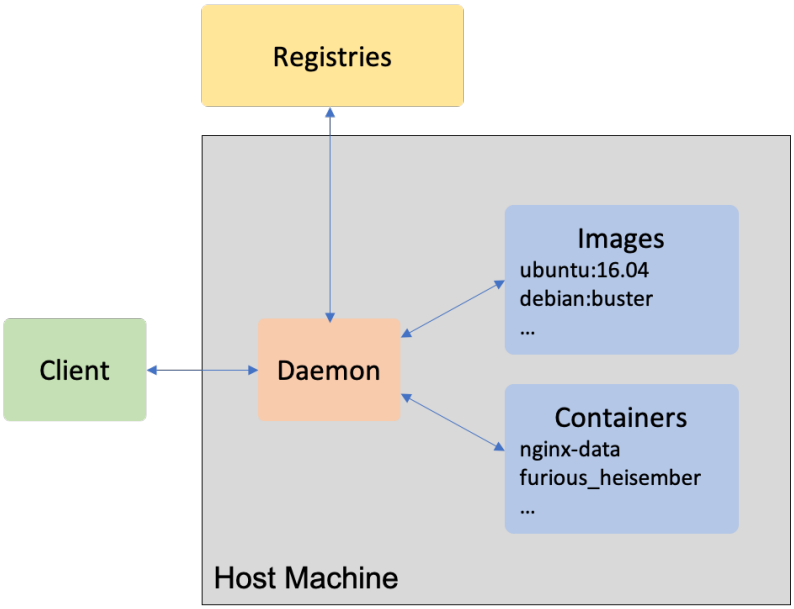


Figure 2.2: Docker architecture.

2.2.2 Key technologies for the Docker platform

The Docker engine uses a proprietary execution environment to create containers, namely the *libcontainer* library. It is crucial for the Docker platform and is tightly tied

to a couple of Linux kernel features:

- *cgroups*. Abbreviation of *control groups*, the cgroups feature limits, accounts for, and isolates the use of resources such as CPU and memory, of Docker containers. It is also responsible for *freezing* and *unfreezing* containers.
- *namespaces*. It partitions kernel resources, separating the container's filesystem, hostname, users, networking and processes from the rest of the system.

The libcontainer library leverages these features mainly for ensuring container isolation and resource control.

Another important technology used by Docker is the *Union File System* (UFS), which allows to store the layers for containers. The UFS can be provided by diverse storage drivers, such as *AUFS* and *devicemapper*.

2.2.3 Dockerfiles

The first step to run a Docker container is the creation of a Docker image. As mentioned in Section 2.2, the Docker Hub contains plenty of ready-to-use container images that prevent the users from creating their own images. This is particularly helpful for common application software such as databases; users can download existing images and simply add (if needed) their configuration files and/or data.

When a user does not find any official image that suits his needs or he wants a container to host his own application, he needs to create an image. The primary way to create a new Docker image is through the composition of a *Dockerfile*. A Dockerfile is a simple text file containing the set of instructions that will be run by the Docker daemon to create the image. Below, we provide a non-exhaustive list that aims at summarizing the main Dockerfiles instructions. Docker instructions shall be written in upper case letters.

ADD - It copies files from a local path or remote URLs into the image. If archives are added, they are automatically unpacked. A simpler command similar to ADD (which has many functionalities) is COPY.

CMD - It allows to specify a command to run when the container is started. If the instruction ENTRYPOINT has been defined, the CMD command will be interpreted as an argument to the ENTRYPOINT. There can be one CMD instruction in a Dockerfile. If more than one is present, only the last CMD will take effect.

COPY - Another possibility to copy files from a local path or a URL into the Docker image. It is a simple instruction that requires two arguments: *src* and *dest*. It copies the file or the directory at *src* to *dest*, inside the image.

ENTRYPOINT - It allows to set an executable that will be run when the container starts. Analogous to the CMD instruction, only the last ENTRYPOINT will have effect.

EXPOSE - Used to specify to Docker that the container built on this image will have a process listening on a given port (or ports). The EXPOSE instruction is a way provided by Docker to make containers accessible from the outside world.

FROM - It is the first instruction in a Dockerfile and sets the *parent image* of the container. The image is specified as `IMAGE:TAG` (e.g., `debian:buster`) and all the subsequent instructions will be run on top of this image.

RUN - Instruction used to run Linux commands while creating the image.

VOLUME - It allows to define a specific file or directory as volume. It is possible to define multiple volumes.

WORKDIR - Used to set the working directory for the subsequent instructions. It can be used multiple times within the same Dockerfile.

2.2.4 Docker commands

The Docker commands allow to create images from the Dockerfiles, and create, manage and monitor the containers. There exist several commands to perform such actions, and below we provide a brief overview on some of them.

`docker build` - Command to build the Docker image from a Dockerfile. This operation may take time, it depends on the complexity and the number of instructions present in the Dockerfile. Once the command is finished, the new image can be used and will be listed by the `docker images` command.

`docker create` - It is similar to the `docker run` command, with the difference that it creates a container from an image but does not start it. It is possible to start the container with the `docker start` command.

`docker ps` - It lists the containers and reports high-level information, such as the container's ID and status. By default it shows only the running container but the argument `-a` allows to get information of all containers (including the ones with status "created", "paused", and "exited").

`docker rm` - Command that removes containers. By default is not possible to remove a running container, unless the argument `-f` is used.

`docker run` - It is the most complex Docker command and allows to create a container on top of an image and start it. This command supports several arguments, which allow users to configure how the image is run, override Dockerfile settings, and set privileges and resources for the container.

`docker start` - It allows to start a container whose status is "exited" or "created". In the former case the container was stopped with the `docker stop` command, in the latter, it was only created and never started.

`docker stop` - With this command, the main process inside the container will receive a `SIGTERM` signal and the container is stopped changing its status to "exited".

2.2.5 The Docker overhead

The Docker overhead is the runtime performance cost to run Docker containers. It is due to several processes whose number varies depending on different factors, such as the number of containers running in background or foreground. The Docker processes

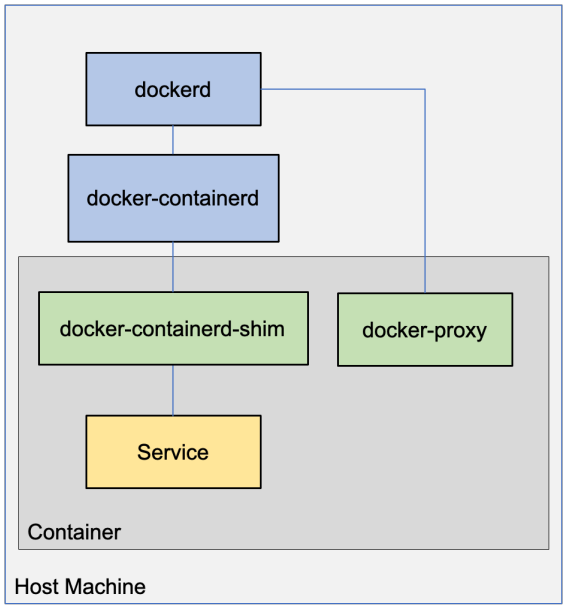


Figure 2.3: Diagram of the Docker processes tree.

are divided into persistent and non-persistent processes. The former are always active, no matter whether there are running containers or not; the latter are created as soon as a Docker container is started. Figure 2.3 shows the Docker processes tree.

The persistent processes are two:

- `dockerd`. It is the Docker daemon process (described also in Section 2.2.1). It is in charge of managing the Docker images and containers, as well as the entire Docker platform.

- `docker-containerd`. It is in charge of managing the containers on the Docker host, *independently* of the Docker daemon. In order to do this, it uses a particular non-persistent process, i.e., the `docker-containerd-shim` process. `docker-containerd` has been introduced to avoid disruption in the normal container execution while the Docker daemon is performing specific maintenance actions, such as rebooting or upgrading.

The non-persistent processes, instead, are:

- `docker-containerd-shim`. It is created by the persistent process `docker-containerd` to facilitate some crucial operations on the Docker container. Firstly, it allows runtime, low-level components, to exit after the container is started. This avoids the execution of long-running runtime processes for the container management and allows to isolate the dockerized application and the shim process. Secondly, it can keep the standard input-output (STDIO), the standard error (STDERR) and other file descriptors open for the container. Finally, it reports back to the Docker daemon the "exit" status of the container when it is terminated.

- `docker-proxy`. For each port exposed by a container, a `docker-proxy` process is created. It operates in user space and for each packet received at the specified host port, redirects it to the container port. This process was introduced to get around a limitation of the old kernels. Nowadays it is not used but kept only for backwards compatibility purposes.

- `docker`. Process in charge of managing the container user interface. If the container runs in background, this process is not created.

2.3 Performance evaluation of the Docker overhead

While VMs are a robust and consolidated technology, the container-based solution has only in the last years become appealing, thanks to the emergence of Docker. Docker is considered a lightweight solution with a low overhead when virtualizing services. Taking into account this, we measured and characterized the Docker overhead implied by the implementation of two applications within containers.

For our analysis, among the multitude of Internet services, we selected video streaming and multiplayer gaming, due to their steadily increasing growth and impact. Indeed, the video streaming is one of the biggest contributors to Internet traffic: it generates a greater amount of data than a picture or a web page, and secondly, countless videos are

daily streamed on the Internet (YouTube videos, Netflix series, in addition to videoconferencing tools such as Skype or Google Hangouts). At the same way, the online gaming industry is continuously on the rise too, with increasing investment in development of new applications and technologies. In addition to their popularity, video streaming and gaming represent service models with profoundly different requirements. A video server implies a low computational effort but generates a fairly large amount of data. A gaming server, instead, requires a much higher CPU resources and generally generates a lower traffic (which depends on the level of interaction between the players in the gaming session), traveling in both directions.

This section hence provides details on the testbed used to collect data and presents the obtained experimental results.

2.3.1 Testbed description

To investigate the overhead cost of providing the video streaming service and the online gaming through Docker containers, we took an experimental approach and deployed the testbed shown in Figure 2.4. We considered a MEC scenario where the containerized

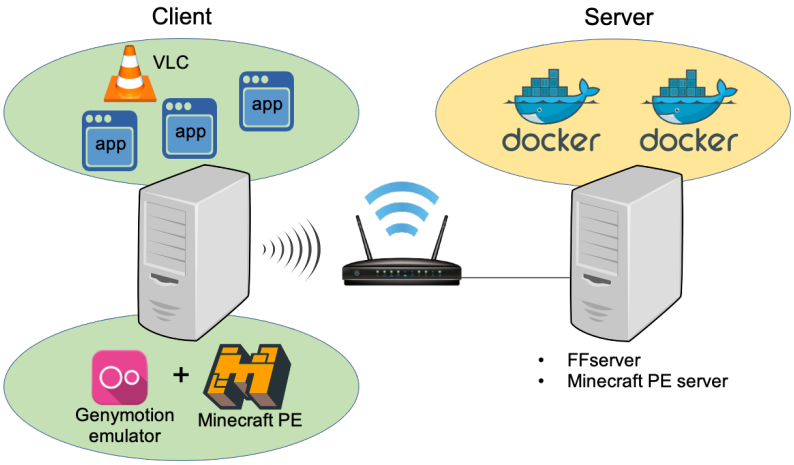


Figure 2.4: Layout of our testbed: clients access the containerized servers through a Wi-Fi AP.

services are provided by servers deployed at the edge of the network infrastructure. For the sake of simplicity, we used the Wi-Fi as wireless access technology. However, it is possible to replace it with any other technology (e.g., LTE), without altering the architecture of the testbed.

The video server selected for our experiments is *FFserver*[1] [42], part of the *FFmpeg* framework [41]. It streams both live and non-live audio and video contents over the

[1]While FFmpeg is still an active project, the FFserver command-line program was removed. The last version of FFserver is 3.4; the version used in our tests is 3.1.3.

Internet. FFserver supports clients running on multiple platforms, including mobile devices such as tablet and smartphones. At client side, instead, we used *VLC* [104] as video player, which ran on a Linux host.

As far as the gaming server concerns, we used *Minecraft Pocket Edition*[2]. The clients supported by this server are only mobile devices, thus we resorted to an Android emulator. We chose *Genymotion* [47], a powerful virtualization platform capable of emulating many Android devices and compatible with Linux systems. Unlike the streaming of a video, the gaming case requires the interaction of a human being. This is a problem when plenty of tests with several clients are performed. To address this need, the Android application *FRep* [45] was installed in each emulator. With this tool, it is possible to record a sequence of taps on the screen and then replicate them over each emulator. In this way, the interaction player-game was automated.

2.3.2 Measuring the Docker overhead

In our testbed, the dockerized servers ran on a Linux Ubuntu desktop. On these systems, the CPU consumed by any process can be monitored by retrieving the unique identifier of the process (i.e., PID) and then parsing the file `/proc/PID/stat`. In particular, two fields contain information about the CPU consumption: the *user time* (called `utime`) and the *system time* (called `stime`). The former is the time length for which a process has been scheduled in user mode, the latter is the time the process spends in system mode. The values of the two fields are expressed in *CPU ticks*. Generally, in Linux, 1 CPU tick corresponds to a CPU usage of 10 ms [32]. To have a complete overview of the whole CPU usage evolution over time, we sampled the `stat` file of each process described in Section 2.2.5 once per second.

An important element that could affect the Docker CPU consumption is the data sent and received by each container. In order to monitor each byte processed by the network card(s) of a host, Linux systems provide the file `/proc/net/dev`. This file contains, for each network card of a host, a lot of information, such as the number of packets sent and received as well as the quantity of bytes transmitted and received. Since Docker creates a virtual ethernet card for every container, it is easy to monitor the data traffic managed by each container. Therefore, also in this case, we sampled the `dev` file once per second.

2.3.3 Experimental results

The machine used to virtualize the two services was a desktop with 32-GB RAM memory, an octa-core *Intel Core i7-4790 @ 3.60GHz* processor, and running Ubuntu OS. We performed many tests, each one 300s-long. In every test involving the video service,

[2]We used the Minecraft Pocket Edition version 0.10.5.

the video streamed by the dockerized server instances was always the same, which had the following features: *(i)* mpeg format with *(ii)* an average bitrate of 4215 kbps and *(iii)* resolution 1280x720 (720p).

In the video streaming use case, we used a single machine as client, as depicted in Figure 2.4. On the contrary, for the gaming case, up to two different Linux machines were used at client side since Android emulators are computationally heavier than VLC instances. Both the client and the server machines were connected under the same subnet and associated to the Wi-Fi AP. The images developed to build the two containers are available in the Docker Hub[3][4].

Below, we present the results obtained through the experimental tests. First, we fixed the number of clients to 8 and varied the number of dockerized servers, while then we did the opposite, i.e., we considered a single server and varied the number of clients.

Fixed number of clients

This first set of experiments is divided into four different case studies. Each one differs for the number of dockerized servers, i.e., 1, 2, 4, and 8, whereas the number of clients is fixed to 8. As mentioned above, each dockerized FFserver streamed the same video, which was stored in every container.

Figure 2.5 shows the Docker overhead CPU consumption for both video streaming and gaming, along with the results for the case with no running containers. The figure shows important aspects, both for the video streaming service and the online gaming. As far as the video server is concerned, it can be noticed that the CPU consumption of the persistent processes (i.e., `dockerd`, `docker-containerd`) is not affected by the number of running containers. Indeed, the amount of CPU used by these two processes together is around 40 ticks (throughout the 300 s of each test) regardless the number of running containers. In addition, the `docker-containerd-shim` processes do not consume CPU. It is important to notice that the only non-persistent process that we can observe is `docker-containerd-shim`. Indeed, the `docker-proxy` was switched off since our Docker host had a recent kernel version[5], whereas the `docker` process, which manages the container in foreground, cannot be observed because our containerized services ran in background. If no non-persistent processes run, then the only cost to dockerize the video server instances is due to the creation and termination of the containers. For the online gaming, instead, we observe a different behavior because both the persistent and the non-persistent processes consume CPU. In particular, the CPU utilization of all the processes depends on the number of running containers.

[3]Video server: https://hub.docker.com/r/giuseppeav/ffmpeg/.

[4]Gaming server: https://hub.docker.com/r/marcomali/minecraft/.

[5]The Docker host kernel version is 4.4.

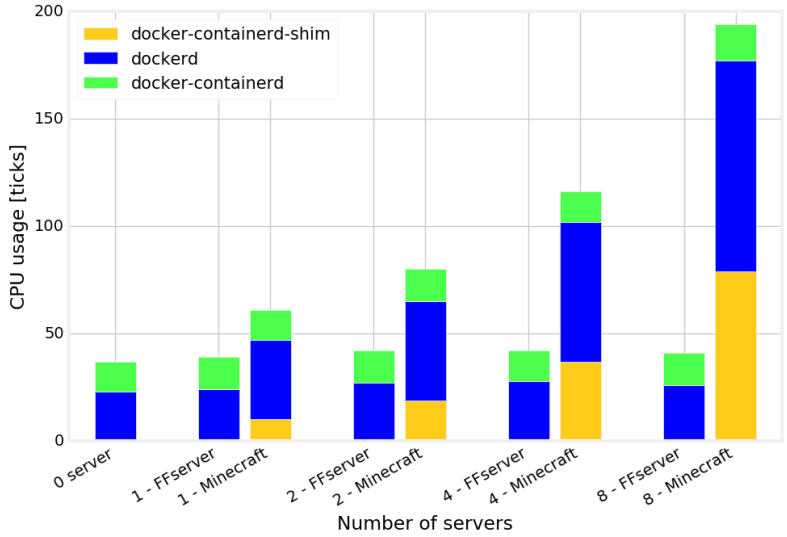
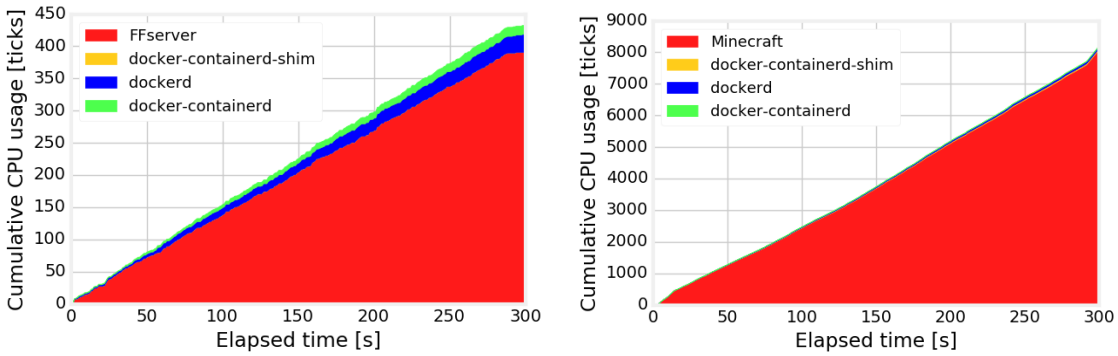


Figure 2.5: Docker overhead for 8 clients and a varying number of servers.

In the two use cases, the diverse CPU utilization by the non-persistent processes can be related to different behaviors of the applications. A significant example is represented by the logging operations performed by the two servers. Indeed, we must take into account a couple of aspects: first, the `docker-containerd-shim` process, among its other tasks, manages the STDIO and STDERR streams; second, the Minecraft game server produces much more logs than FFserver. Together these two observations partially explain why the shim process of each Minecraft server is subject to a higher workload, and therefore to a higher CPU consumption.



(a) FFserver.

(b) Minecraft.

Figure 2.6: Temporal evolution of the CPU consumption due to the application and to the Docker overhead, in the case “8 clients - 4 servers”.

The temporal evolution of the CPU consumed by the Docker processes and by the dockerized applications is depicted in Figure 2.6. The plot refers to the specific case

with 8 clients and 4 servers. First of all, we can note that the video streaming server consumes much less CPU, in the order of 1/20th than the gaming application. As we already highlighted, this is due to the fact that a video server does not require high computational capacities, because it simply reads and forwards data. The second observation concerns the Docker overhead, which is very low in both cases. However, although the virtualization cost of the video server is much lower than the one of the gaming server, the overhead of the latter is negligible with respect to the application CPU load, while it accounts for over 10% in the case of video streaming.

Fixed number of servers

In this second experimental campaign, we used a single server and varied the number of clients between 1 and 8. Figure 2.7 shows the CPU utilization due to the Docker overhead for both the video streaming and the online gaming. For what concerns FF-

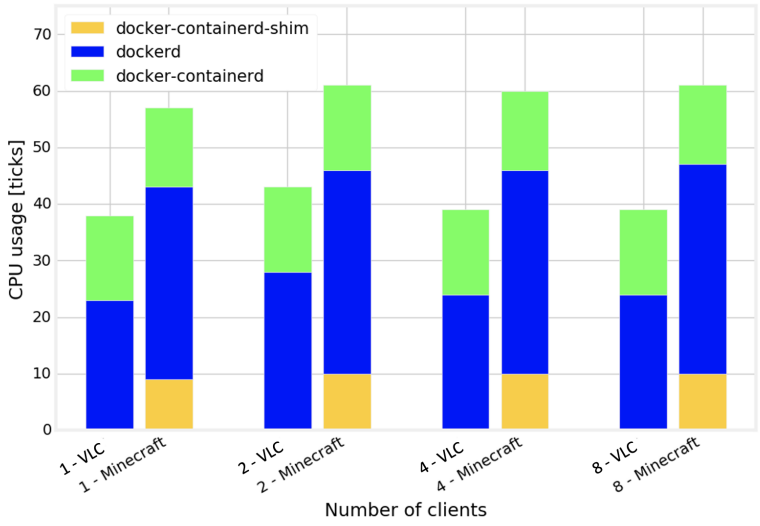


Figure 2.7: Docker overhead CPU consumption for a single server and different number of clients.

server, we can once again observe that *(i)* the `docker-containerd-shim` process does not consume CPU, and *(ii)* the CPU utilization for both persistent processes is not affected by the server workload (in this specific case, independent of the number of clients served). The same behavior can be noticed in the Minecraft use case, indeed the CPU usage by the two Docker persistent processes and the `docker-containerd-shim` is always constant. These experimental results demonstrate that the Docker overhead, at least for the two services selected, is not dependent on the workload of each container.

An interesting common point between the two case studies and the different types of tests is the CPU consumed by the persistent process `docker-containerd`, which is

always equal to 0.05 CPU ticks per second. Moreover, we saw that this process was not crucial in our tests. Indeed, it performs specific maintenance actions, (e.g., rebooting) on the Docker daemon, thus it is unrelated to the normal container execution. The termination of `docker-containerd` hence does not affect the behavior of the running containers.

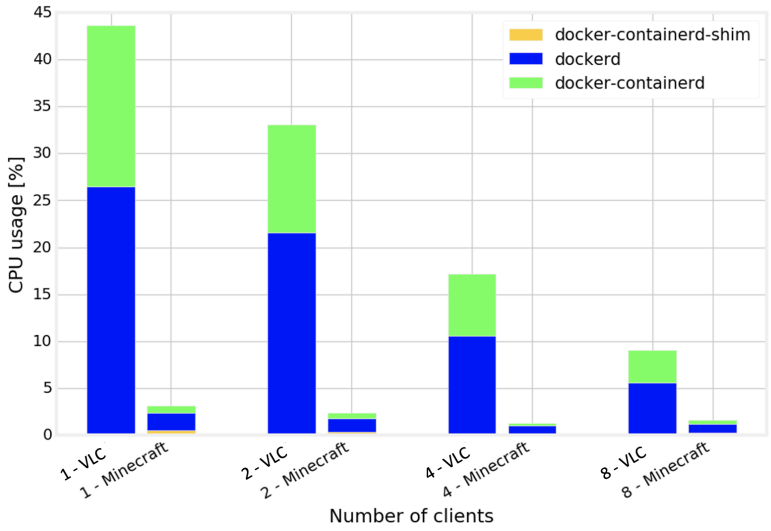


Figure 2.8: Percentage of CPU consumption due to the Docker overhead, for one server and diverse number of clients.

Finally, Figure 2.8 shows the overhead impact on the total CPU consumed by the dockerized services. As mentioned above, the CPU consumption of the video server is much lower when compared to the one of the gaming. For this reason, the Docker overhead has a greater impact on the video streaming use case. In particular, with a single client, it accounts for almost 45% of the total CPU consumed by the containerized server. However, since the higher the number of clients served, the larger the CPU consumption of the application itself, the overhead impact steadily decreases and falls below 10% for eight clients. The same behavior can be observed for the gaming server, even if less noticeable because the impact of the overhead is already very low even with a single client connected.

Data processed by Docker containers

Last question is whether the data processed by containerized services affects the Docker overhead CPU consumption. Figure 2.9 represents the amount of data transmitted and received by the two dockerized services, in the case study with 4 clients and 1 server. As expected, FFserver processes much more data than the Minecraft server, indeed it transmits to four clients a video with a size of around 160 MB. Other two facts are worth underlining. First, we can infer that FFserver does not buffer (it is used mainly

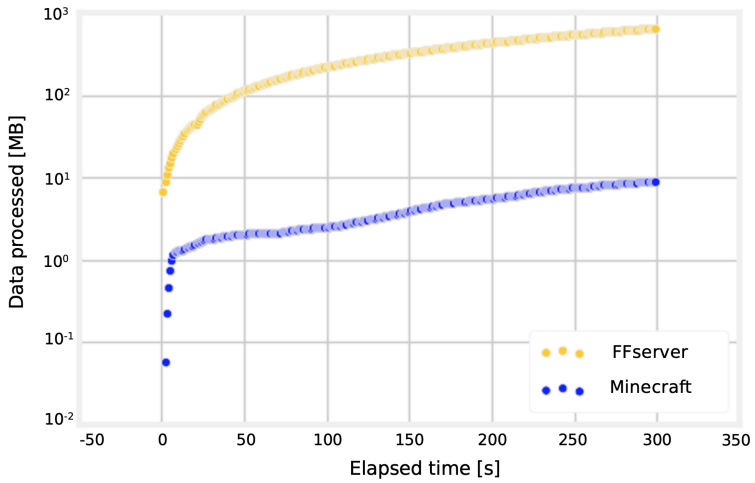


Figure 2.9: Megabytes of data processed by the two applications, in the specific case 1 server - 4 clients.

for live streaming), because the amount of data transmitted by the video server increases linearly with time. Second, the Minecraft server exchanges the largest amount of data with the clients in the first time instants. Indeed, at the beginning, the server sends to all players information about the virtual world in which they will play. The other packets are due to in-game interactions between server and clients.

By comparing this plot and Figure 2.7, it is possible to conclude that the data processed by dockerized servers does not affect the Docker overhead. The data transferred by FFserver is about 638 MB in 300 s, while only 8 MB for Minecraft. However, looking at Figure 2.7, we can see that, in the case "1 server - 4 clients" the Docker CPU consumption for Minecraft is much higher.

2.4 Final remarks

The experimental results confirmed that Docker can be employed as a lightweight virtualization solution. We selected two services with opposite requirements in terms of CPU load and generated network traffic. For both services, the Docker overhead ranged between negligible and moderate. The overhead was found to be independent of both the number of served clients and the data processed by the two services.

The purpose of this study was the analysis of the Docker performance, in order to evaluate its suitability within the 5GT platform. Docker exhibited an excellent performance, proving to be a valid alternative to the robust and reliable hypervisor-based technology. However, the 5GT platform is composed by a complex architecture, with several components that interact each other. Due to some incompatibilities between

Docker containers and a few 5GT components, we were forced to select the hypervisor-based technology to virtualize the network functions of the 5GT architecture.

Chapter 3

Design, Implementation and Performance Analysis of a Collision Avoidance Algorithm through Simulation

As discussed in the previous chapter, the 5G technology is envisioned to expand and improve the existing mobile networks. 5G aims at ensuring ultra-low latency, extensive coverage, ultra-high reliability and high data rate. As a result, many industries have a growing interest in reaping the benefits of 5G networks, in order to increase the quality of their existing services or to deploy new ones. Indeed, as shown in Table 2.1, innovative vertical services can demand very strict and diverse requirements, which cannot be guaranteed by legacy 3G and 4G networks.

In this challenging context, the 5GT project has designed a NFV-based 5G mobile transport and computing platform able to meet the specific needs of a wide range of services. This project has been driven by some relevant vertical industries, from various sectors such as automotive, entertainment, e-Health and e-Industry. Such industries are some of the most affected by the deep technologies changes of the last two decades. For instance, the automotive industry has the possibility to provide safety services able to dramatically reduce (aiming at zero) fatalities on the road, whereas the media and entertainment industry is called to satisfy the huge demand from users of media-rich contents and provide a better quality of experience. These service requirements pose a great challenge to the network infrastructure [5].

Within the 5GT project, each of the vertical industry has developed a use case (UC), in order to assess the 5GT infrastructure and architecture. Among these UCs, as mentioned in Chapter 1, we have focused on the automotive one, which consisted in the development of a cellular-based vehicle collision avoidance (CA) service, based on an ad-hoc CA algorithm.

In this chapter, we first present an overview of the main services and applications

involving the automotive field, and the vehicular communication technology necessary for their development. Then, we describe the design and the implementation of the CA algorithm. Finally, we discuss the simulation testbed used to simulate the service and evaluate its performances.

3.1 The automotive use cases

The automotive industry is currently undergoing many technological transformations. Indeed, a growing number of vehicles are equipped with communication hardware, which enables access to the Internet and connectivity with third parties. This permits the development of innovative automotive services, to offer a variety of applications, such as entertainment, safety and many others. Nevertheless, for their implementation, the 5G technology is strictly necessary to accomplish the reliability and delay constraints they require.

Vehicular applications cover a wide variety of potential consumer needs and business model. In the literature, it is common to group UCs together according to their purpose and requirements. As a result, vehicular applications can be classified in four categories [61]:

1. *Infotainment*: Infotainment applications provide informative or entertaining services such as instant messaging and delivery of geo-specific advertisements, to drivers and passengers. These applications does not require neither ultra-low latency (500-1000 ms) nor high data rate (60-80 Mbps) [10].

2. *Traffic Efficiency*: This kind of applications aims at optimizing the flow of road traffic and increasing the driving experience. For instance, an on-board GPS able to find the best route according to the traffic conditions is a concrete example. The requirements of these services fall between the traffic safety and infotainment categories [35]; overall, they demand low latency and a high reliability [61].

3. *Traffic Safety*: The objective of these applications is to reduce the risk of accidents and human casualties. The exchange and processing of messages and data between vehicles and vulnerable road users (VRUs), e.g., pedestrians and bikers, enables the deployment of a variety of services, targeted at alerting road users in case any potentially dangerous situation arises. Such applications mainly require ultra-low latency and ultra-high reliability [35, 82]. However, all those services relying on remote processing for real-time event handling may need high-throughput data transmission, in the order of hundreds of Mbps. For instance, an application for the road sign and obstacle recognition, require up to 700 Mbps of throughput [28].

4. *Cooperative Driving*: These applications can be both classified as *traffic safety* applications and form a fourth category. They are characterized by strict requirements and are uniquely suited to autonomous vehicle operation. For instance, cooperative platooning is a cooperative driving application requiring a quite low data rate but an ultra-low latency, between 2 and 10 ms [40].

3.1.1 Vehicle-to-Everything communication

The vehicular services and applications described in Section 3.1 require that cars are connected and able to communicate between each other, with other road users or with the infrastructure. The data exchange between a vehicle and the entities surrounding it (e.g., other vehicles, pedestrians) is called *vehicle-to-everything* (V2X) communication.

The third generation partnership project (3GPP) standardized the support for V2X communications, as part of the Long Term Evolution (LTE) technology, in Release 14 [1]. Figure 3.1 represents all the supported types of communications:

- **Vehicle-to-Network** (V2N): this is the communication between the vehicle and servers or cloud-based services, which are reachable through the cellular infrastructure, i.e., the base station (BS).
- **Vehicle-to-Infrastructure** (V2I): this kind of communication involves the transmission of data, within a certain radio range, between vehicles and road side units (RSUs). The RSU is usually co-located within a BS or in standalone devices, such as traffic lights.
- **Vehicle-to-Vehicle** (V2V): this kind of communication involves a direct data exchange among vehicles, in radio range between each other.
- **Vehicle-to-Pedestrian** (V2P): it represents the direct communication between vehicles and VRUs.

In Release 14, 3GPP standardized both the direct communications between user equipments (V2V and V2P), and the V2I/V2N communications: the former leverage the sidelink (PC5) interface, whereas the latter occur over the cellular LTE-Uu interface, in the traditional licensed spectrum. Furthermore, the direct communication supports two diverse communication modes: Mode 3 and Mode 4. In Mode 3, vehicles can communicate only if under coverage of a BS, as the allocation of radio resources is controlled by the network. As far as Mode 4 is concerned, instead, the resources are pre-configured, in such a way that vehicles do not need any cellular network coverage to communicate [27].

Further specifications for LTE-V2X communications are detailed in Release 15 [2], which provides the service requirements to support several V2X scenarios, such as vehicle platooning and extended sensors. Finally, Release 16 [3] contains enhancements to the 5G architecture to meet the most demanding V2X performance requirements. At

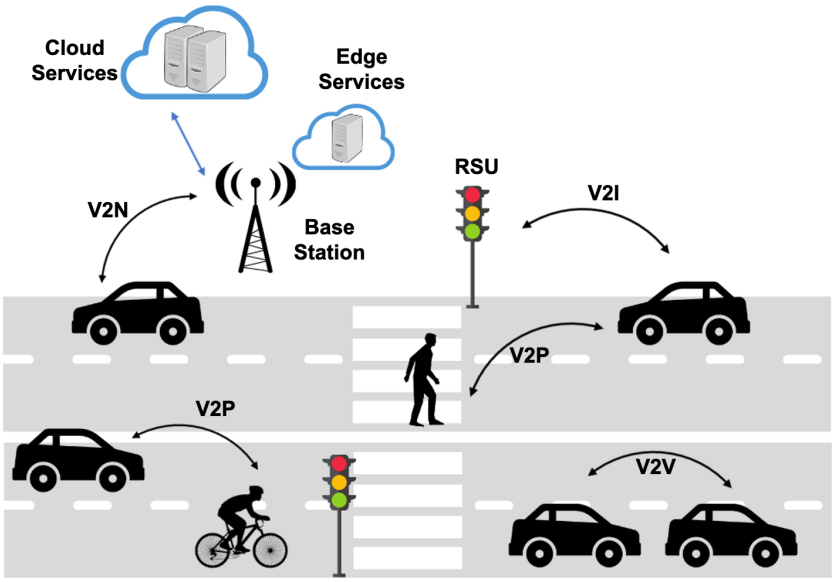


Figure 3.1: V2X communication modes and entities.

the time of writing, Release 16 is at its draft stage. The smooth evolution from LTE-V2X to 5G V2X is commonly referred to as cellular-V2X (C-V2X) [11].

Alternative access technologies

The usage of cellular network is not the only solution enabling vehicular communication. For nearly two decades, IEEE 802.11p (a particular amendment of IEEE 802.11), has been investigated as radio access technology for V2V and V2I communications [22, 57]. The main advantage of 802.11p is that it ensures low-latency connectivity among vehicles even in absence of a roadside infrastructure. Although it offers good performances in low congested traffic scenarios, it seems to suffer from throughput degradation and large delays when the number of vehicles increases significantly [71, 70]. These limitations are mainly due to its very basic physical layer and the absence of a mechanism able to reduce the collisions, which is particularly critical for broadcast communications under congestion [24]. An exhaustive comparison between 802.11p and C-V2X, as well as a detailed description of the two technologies, are out of the scope of this thesis but widely covered in the literature [13, 23, 21, 24]. Overall, the choice of the best communication technology is still subject of an intense debate in the scientific community.

Recently, also the millimeter wave (mmWave) band has been investigated as access technology for automotive applications. mmWave communications ensure a high data rate up to several Gbps [75] by utilizing the Extremely High Frequency (EHF) band occupying the 30-300 GHz band of the radio frequency spectrum. Such an access technology is particularly appealing both for V2V communications between very close vehicles

(e.g., to support cooperative sensing in a high-density platoon) and for V2I communications, for offloading massive amounts of data (on-board sensors may generate a large amount of data, in the order of terabytes per driving hour [28]). Despite a very high throughput and low latency, mmWave communications suffer a severe isotropic path loss and require a Line-of-Sight (LoS) connection between the transmitting and receiving node, which is difficult to achieve in mobile scenarios [61]. Further information on the use of mmWave band for C-V2X applications can be found in [98, 50, 49]. In general, mmWave is not yet a mature technology for vehicular networks but it has the potential to meet some of the boldest requirements of next-generation transportation systems, being able to provide excellent performance in terms of data rate and latency, both in urban and high-mobility highway scenarios [48].

3.1.2 The vehicle collision avoidance service

The *vehicle collision avoidance* service was selected as an automotive safety UC in the context of the 5GT project. The aim of the service is to warn drivers about any imminent danger that may result in a collision (e.g., the presence of an unseen vehicle), and eventually activate the emergency braking system to avoid the accident. In order to exploit this service, vehicles should be equipped with on-board units (OBUs) through which they can send and receive data.

The vehicles periodically transmit anonymous *Cooperative Awareness Messagges*[1] (CAMs), which contain kinematic and dynamic data of sender, such as position, speed, acceleration and heading. The CAM generation follows a dynamic scheme standardized by ETSI [33]. According to this scheme, the higher the variation of vehicle's speed, position or heading angle, the higher the CAM frequency will be. ETSI also defined the minimum and maximum frequency values: 1 Hz and 10 Hz, respectively [33].

The service can leverage both the V2V and V2I communication modes. In the first case, the CAMs generated by a vehicle are sent as broadcast packets and received by any other car in its close proximity. Each vehicle equipped with a unit running a CA algorithm can autonomously foresee potential dangerous situations and warn the driver. In case of V2I communication, instead, a centralized approach is used, with a single instance of the CA algorithm, running in a server in proximity of a BS. CAMs are hence transmitted toward this server, which, in case of danger, generates and sends an alert to the vehicles involved. The alert is encoded inside a *Decentralized Environmental Notification Message* (DENM), which can simply warn human drivers, or activate the autonomous emergency braking system.

In order to increase the accuracy of the service, the data generated and received by each vehicle can be processed together with additional information, collected thanks to Advanced Driver-Assistance Systems (ADAS), by relying on on-board sensors.

[1] Equivalently, the *Basic Safety Messages* (BSM) standardized by SAE could be considered.

3.2 The collision avoidance algorithm

The core of our safety service is the collision avoidance algorithm, presented in Algorithm 1. With a different flavor, the basics of the algorithm have been used in [69], which presents a top-down and specification-driven design of an adaptive peer-to-peer collision warning system. Note that, although the safety service we want to develop for the 5GT project focuses on collisions between vehicles, since the algorithm is based on generic trajectories, it can be applied to detect collisions between any kind of entity (e.g., between vehicles and pedestrians).

The algorithm, which runs at each new CAM message received, requires as input: *(i)* the position $\vec{p}_0$, speed $\vec{v}$ and acceleration $\vec{a}$ of the vehicle transmitting the CAM; *(ii)* the set $\mathscr{B}$ containing the information of all other vehicles traveling in the same area. As soon as a CAM is received, the algorithm first initializes the set $\mathscr{C}$ of nodes with which the sender v of the CAM could collide (Line 1), then evaluates its position for each future time instant (Line 2 and 3). At this point, the algorithm reads position, speed and acceleration of each entity that recently sent a CAM (Line 5), and calculates

Algorithm 1 Collision avoidance pseudocode

Require: $\vec{p^0}, \vec{v}, \vec{a}, \mathscr{B}$

1: $\mathscr{C} \leftarrow \varnothing$
2: $p_x(t) \leftarrow p_x^0 + v_x t + \frac{1}{2} a_x t^2$
3: $p_y(t) \leftarrow p_y^0 + v_y t + \frac{1}{2} a_y t^2$
4: **for all** $b \in \mathscr{B}$ **do**
5: **read** $\vec{\hat{p}}^0, \vec{\hat{v}}, \vec{\hat{a}}$ from b
6: $\hat{p}_x(t) \leftarrow \hat{p}_x^0 + \hat{v}_x t + \frac{1}{2} \hat{a}_x t^2$
7: $\hat{p}_y(t) \leftarrow \hat{p}_y^0 + \hat{v}_y t + \frac{1}{2} \hat{a}_y t^2$
8: $D(t) \leftarrow (p_x(t) - \hat{p}_x(t))^2 + (p_y(t) - \hat{p}_y(t))^2 =$
$= \left[p_x^0 - \hat{p}_x^0 + \left(v_x - \hat{v}_x\right) t + \frac{1}{2}\left(a_x - \hat{a}_x\right) t^2\right]^2 + \left[p_y^0 - \hat{p}_y^0 + \left(v_y - \hat{v}_y\right) t + \frac{1}{2}\left(a_y - \hat{a}_y\right) t^2\right]^2$
9: $\mathscr{T} \leftarrow t : \frac{\mathrm{d}}{\mathrm{d}t} D(t) = 0$
10: **for all** $t^\star \in \mathscr{T}$ **do**
11: **if** $t^\star < 0$ **or** $t^\star > t2c_t$ **then**
12: **continue**
13: **end if**
14: $d^\star \leftarrow \sqrt{D(t^\star)}$
15: **if** $d^\star \leq s2c_t$ **then**
16: $\mathscr{C} \leftarrow \mathscr{C} \cup \{b\}$
17: **break**
18: **end if**
19: **end for**
20: **end for**
21: **return** $\mathscr{C}$

their future positions (Line 6 and 7). In Line 8, we compute the square of the Euclidean distance between v and the generic entity b.

Since we are interested in the minimum value of $D(t)$, in Line 9 we compute the set $\mathscr{T}$. This computation represents the most complex part of the algorithm because requires solving a 4^{th} grade equation. Each value $t^\star$ of the set $\mathscr{T}$ is defined as time instant at which the distance between the two entities is minimum. For each $t^\star$, the algorithm checks if it is included between 0 and a threshold $t2c_t$ (*time to collision* threshold). If $t^\star$ is negative, the two entities are getting farther apart, whereas, if $t^\star$ is greater than $t2c_t$, we assume that the possible collision is not imminent. In both cases, no action is performed (Line 11). On the contrary, if $t^\star$ is between 0 and $t2c_t$, the algorithm computes the distance $d^\star$ at which the two entities will be at time $t^\star$ (Line 14). Such a distance is compared against a second threshold called $s2c_t$ (*space to collision* threshold): if $d^\star$ is greater, the algorithm moves to the next iteration of the *for* cycle, otherwise b is added to $\mathscr{C}$ because a collision is deemed likely (Line 16). In this second case the nested *for* cycle in Line 10 is interrupted (Line 17) and the algorithm moves to the next entity.

After processing all the CAMs in $\mathscr{B}$, the algorithm returns the set $\mathscr{C}$ containing the entities with which v is on a collision course. The set $\mathscr{C}$ could be empty, in that case no action is taken. Otherwise, an alert message (i.e., a DENM) is generated and sent to v as well as to all the entities in the set $\mathscr{C}$.

3.3 Simulation testbed and methodology

The CA service developed for the 5GT project is based on the algorithm presented in Section 3.2. To assess its effectiveness and reliability, we used the SimuLTE-Veins framework [90], integrating our CA service inside the simulator. In this context, the CA service leverages the cellular-V2I (C-V2I) communication technology [101], meaning that the collision detector is hosted in a dedicated server, able to monitor the whole urban area under study. All the CAMs are sent by the vehicles toward the BS, through which they reach the CA server. The latter, in case of danger, alerts the involved drivers by generating and transmitting DENM messages.

There are several works in the literature that are related to safety applications in the automotive domain (e.g., [46]). Many of these works, such as [86] and [108], propose CA applications that do not leverage any mobile network infrastructure. In particular, [86] focuses on collisions between vehicles and pedestrians in industrial plants. In this case, positioning is achieved using a combination of GPS, Micro-Electro-Mechanical Systems (MEMS) and smart sensors, while the type of wireless communication to the control center is not specified. In [108], White et al. propose a way to automatically detect a collision after it has occurred, using smartphone accelerometers to reduce the time gap between the actual collision and the first aid dispatch. A very interesting work is presented in [53], where Hafner et al. develop and assess through field tests the efficiency of decentralized algorithms for two-vehicle cooperative CA at intersections. The

algorithms proposed leverage the V2V communication technology. Nevertheless, simulation tools, which use CA algorithms and C-V2I communications, are usually neglected in literature. For this reason, we implemented a simulation testbed which allows us to provide several experimental results in order to assess the performance of both our CA algorithm and the LTE-V2I technology in vehicular scenarios.

In this section, we focus on the simulator and on the selected urban scenario. Moreover, we provide some details on the service implementation and the processing of the simulation logs.

3.3.1 Simulation tools

In order to assess the performance of the CA algorithm, we used the OMNeT++ simulator [76], a well-known, widely-used modular simulation framework. More specifically, we used the SimuLTE-Veins framework, where OMNeT++ is combined with:

- SUMO [95, 60]: an open-source, highly portable simulator of urban mobility;
- Veins [99, 91]: an open-source framework enabling the communication between SUMO and OMNeT++;
- SimuLTE [100, 102]: an open-source system-level framework for simulating both LTE and LTE-Advanced (LTE-A) networks within OMNeT++.

We used the following versions of the components mentioned above: SimuLTE v1.0.1, SUMO v0.30.0, OMNeT++ v5.1.1.

3.3.2 Reference scenario

The reference road topology is the urban area depicted in Figure 3.2. It was composed of three roads, crossing at two intersections, a pedestrian lane and three pedestrian crossings. As can be deduced from the presence of pedestrian areas, we also considered VRUs, in addition to vehicles. This was done in order to evaluate the performance of the designed algorithm also in case of possible collisions involving different road users. Indeed, as mentioned in Section 3.2, our solution includes a trajectory-based CA algorithm suitable for any kind of road user. As a result, the CA service detected both the vehicle-with-vehicle collisions, which occurred at the two intersections, and the vehicle-with-pedestrian collisions on the zebra crossings.

Every entity in the area was connected to the cellular infrastructure and subscribed to the CA service[2]. Vehicles were equipped with on-board units which leveraged the Uu interface for C-V2I communications. Pedestrians, instead, carried a smartphone with

[2]We are assuming a penetration rate equal to 1.

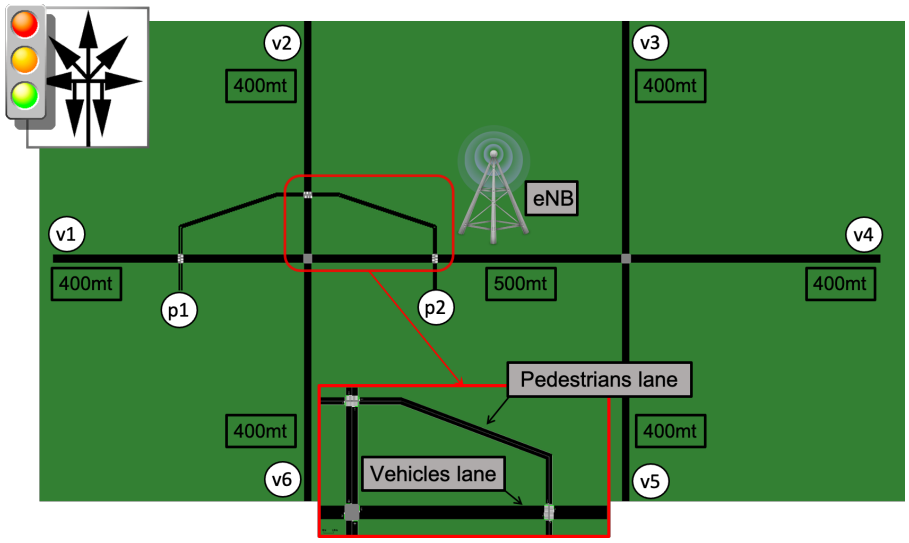


Figure 3.2: Screenshot (from SUMO) of the urban area monitored by the CA algorithm.

cellular connectivity. Both types of entities periodically transmitted CAMs toward the CA server.

The cellular network simulated by SimuLTE-Veins was LTE, therefore the safety service leveraged the LTE-V2I communication technology. The whole considered area was covered by a single evolved Node B (eNB), located at the center of the topology. The server running the CA algorithm can be located at different points of the network infrastructure. In order to study the diverse performances of the service according to its location, we considered two server deployments: at the Metro node (very close to the eNB), namely in a MEC fashion, and in a cloud data-center.

It is worth emphasizing that, even if the selected topology is very simple, it is very widespread, and therefore allows to closely mimic many real-world urban road layouts.

Populating the scenario

We used a realistic mobility model and a realistic generation rate for the simulated vehicles and pedestrians. Vehicles traveled at the maximum speed of 13.89 m/s (i.e., 50 km/h), and followed a straight path, i.e., they did not turn at intersections. Pedestrians moved on the pedestrian lane at a steady speed of 2 m/s. They could cross the street at three different spots. Each generated vehicle was randomly assigned to one of the six entry points at the edge of the scenario (marked as *v1...v6* in Figure 3.2), whereas each VRU was assigned to one of either ends of the pedestrian lane (*p1* or *p2*). Following [58], vehicles and pedestrian arrivals were modeled as Poisson processes with two diverse rates: λ_v for vehicles, λ_p for pedestrians.

In order to have reliable results and a realistic mobility pattern, we performed a stability study on the scenario under consideration. Indeed, it was crucial to set both

the vehicular generation rate λ_v and the pedestrian generation rate λ_p in such a way that the simulated scenario was stable. In this way, the number of vehicles or pedestrians did not grow so high as to yield the following situations:

- the clogging of the intersections due to the presence of too many cars;
- the formation of long queues of low-speed vehicles.

Note that, both situations lead to a low number of collisions that would have not allowed us to properly assess the effectiveness of the CA algorithm.

The dependence of the number of vehicles and pedestrians from the two rates λ_v and λ_p was not linear because of the three spots in which the two entities share the road occupancy (i.e., at the zebra crossings). To properly select the arrival rate of both cars and pedestrians, we simulated the scenario when λ_v varied from 0 to 1.5 (with step 0.05) whereas λ_p took one of five possible values: 0, 0.05, 0.10, 0.15, 0.20. The results for $\lambda_p = 0$ (i.e., without pedestrians) have shown that the value of λ_v for which the average number of simulated vehicles grows linearly with the generation rate (i.e., it is in the stability region) is between 0 and 1.2. As expected, the introduction of VRUs has a significant impact: while with $\lambda_p = 0$ the maximum λ_v allowing stability is 1.2, with λ_p equal to 0.2, only values lower than 0.9 ensure stability. Figure 3.3 shows the growth

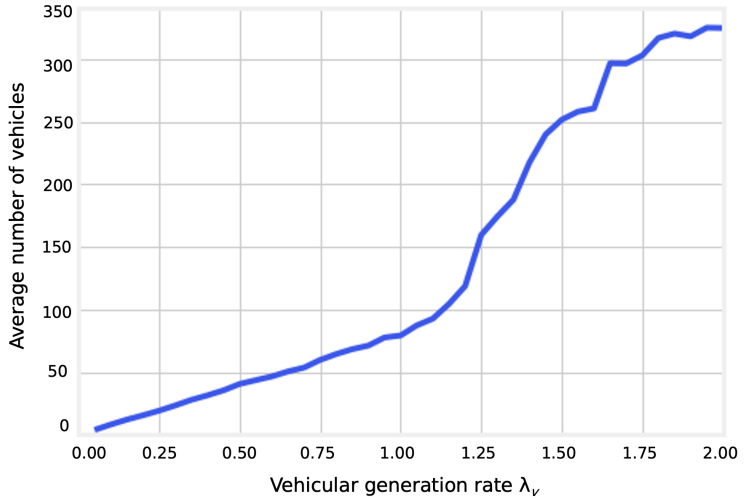


Figure 3.3: Evolution of the average number of vehicles for $\lambda_p = 0.2$ and λ_v ranging between 0 and 1.5.

of the average number of vehicles for different values of λ_v, with λ_p fixed to 0.2.

As a result, for our simulations, we set λ_v to 0.7 and λ_p to 0.1. Furthermore, the value selected for λ_v is consistent with real-world measurements from the city of Turin, Italy [7].

3.3.3 System description

In Section 3.1.2 we provided a general overview of the CA application. Here, we describe in detail how it works within the SimuLTE-Veins simulator, and present our improvements, in order to make the algorithm more efficient.

The frequency at which CAMs are sent by vehicles and pedestrians was different: 10 Hz for vehicles (i.e., 1 CAM every 100 ms), 1 Hz for pedestrians (1 CAM per second). The difference is mainly owing to the speed at which the two entities move. Indeed, considering vehicles traveling at 13.89 m/s, and the same CAM frequency of pedestrians, the error at the server (ignoring the transmission delays) would be in the worst case of 13.89 m. Clearly, an error in this order of magnitude is not acceptable when dealing with safety applications. As a result, the frequency at which vehicles transmitted CAMs was the maximum standardized by ETSI, whereas for pedestrians, the minimum.

Our CA algorithm is able to distinguish between CAMs sent by pedestrians and CAMs sent by vehicles. This gives us two advantages. First, if the CAM is sent by a vehicle, the algorithm looks for possible collisions with both cars and pedestrians, while on the contrary (i.e., with a message sent from a pedestrian), the analysis for pedestrian-with-pedestrian collisions is skipped. The second advantage involves the possibility to set different parameters for the thresholds $t2c_t$ and $s2ct$, according to the type of entity that sent the CAM. This ensures better performances of the algorithm, both in terms of false positives and false negatives (further details in Section 3.4.1).

In order to increase the efficiency and reliability of the service, the CA algorithm implemented in the simulator provided two improvements:

1. Every time the algorithm receives a CAM, it checks if it is up-to-date: if so, the information of the message is stored and the algorithm checks if the sender is at risk of collision; otherwise, the CAM is discarded. In our simulations, the threshold used to determine whether a message is up-to-date or stale (therefore to discard) was different for vehicles and pedestrians: 0.8 s and 4 s, respectively. The diverse values are due to the lower speed at which pedestrians move and thus, even considering older information, the error in the evaluation of the position is smaller.

2. When the algorithm checks if two entities, far from each other, risk to collide, it is loosing time and decreasing its efficiency. To address this issue, we filter some iterations. We therefore introduced a *range of action* and only the entities within the range of action of the sender of the CAM will be checked by the algorithm as potential colliders. The radius varies according to the car speed as follows:

$$Radius = max\{Speed * t2c_t, s2c_t\} \tag{3.1}$$

At this point it should be clear how a CA application works and the implementation of such an application in our simulator. A visual representation of our CA service with the main components is provided by Figure 3.4.

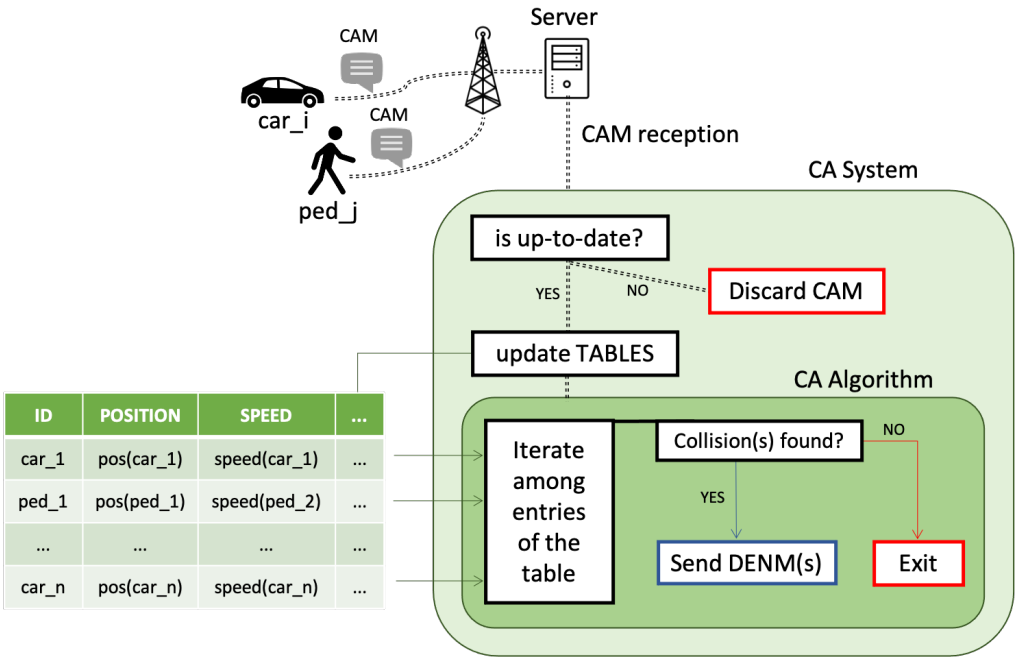


Figure 3.4: Collision avoidance system.

3.3.4 Processing the simulation logs

In order to derive the metrics of interest and evaluate the performances of the algorithm, we collected information from both SUMO and SimuLTE-Veins. In particular, we were interested in:

- position, speed and heading of each vehicle and pedestrian (information contained in the SUMO *Floating Car Data* output);
- each vehicle-with-vehicle and vehicle-with-pedestrian collision occurred (information obtainable from the SUMO error-log file);
- each alert message (DENM) generated by the collision avoidance algorithm, (returned by the SimuLTE-Veins simulator).

Then, through post-processing, we analyzed, *when* each collision occurred and *if* the corresponding DENM was generated by the algorithm. Furthermore, if the DENM was correctly transmitted, we also looked at when it was received, in order to determine if the vehicles had sufficient time to brake and avoid the accident.

Determining if a collision is detected in time

Whether a collision is detected in time or too late is determined in the post-processing phase, by cross-referencing the information collected in the logs of SUMO and SimuLTE-Veins. A collision is considered as "detected too late" when:

$$T_A < T_B \tag{3.2}$$

T_A is the time available to the driver to react to the danger, i.e., the interval between when the driver initiates evasive actions and the actual collision. T_B, instead, represents the time needed by the entity to stop, given its current speed and maximum deceleration. T_A is computed as follows:

$$T_A = T_{FA} - T_D - T_H \tag{3.3}$$

The three elements reported in (3.3) are:

- T_D. It represents the time interval between the moment at which a collision is detected by the algorithm and the moment at which the DENM reaches the driver. It is composed of the transmission time and the processing time. The former includes the time to transfer data from the application server to the eNB, and then to the entities involved in the collision. The processing time, instead, is the time needed at the receiving node to process a DENM, from the time instant at which the first bit is received until the moment at which the human-machine interface (HMI) shows the warning message to the driver. In our simulations, we set the processing time to 400 ms [39].
- T_H. It is the time needed by a human to take action following the prompt of a DENM. This value depends on several variables such as age, travel length, environment, etc. According to [94], we set T_H to 1 s.
- T_{FA}. It represents the time elapsed between the generation of the first DENM related to a possible collision and when such a collision occurs.

Figure 3.5 shows the timeline of the communication between the collision detector and the human driver, highlighting the time intervals discussed above.

3.4 Simulation results

In this section, we first present a sensitivity study on the *time* and *space* thresholds, then we show and discuss our simulation results.

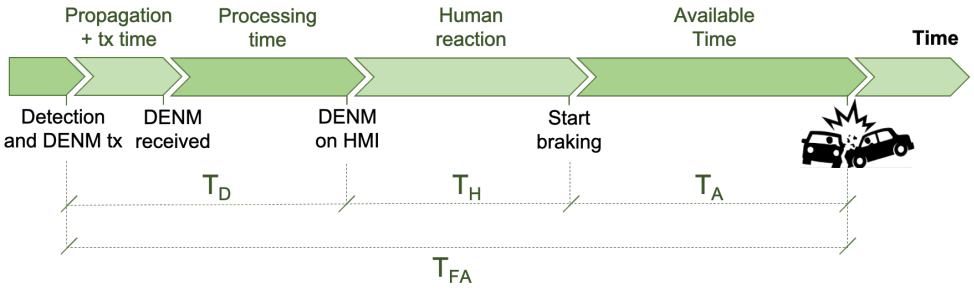


Figure 3.5: Timeline of the communication between the CA server and the vehicle.

3.4.1 Sensitivity study on the collision thresholds

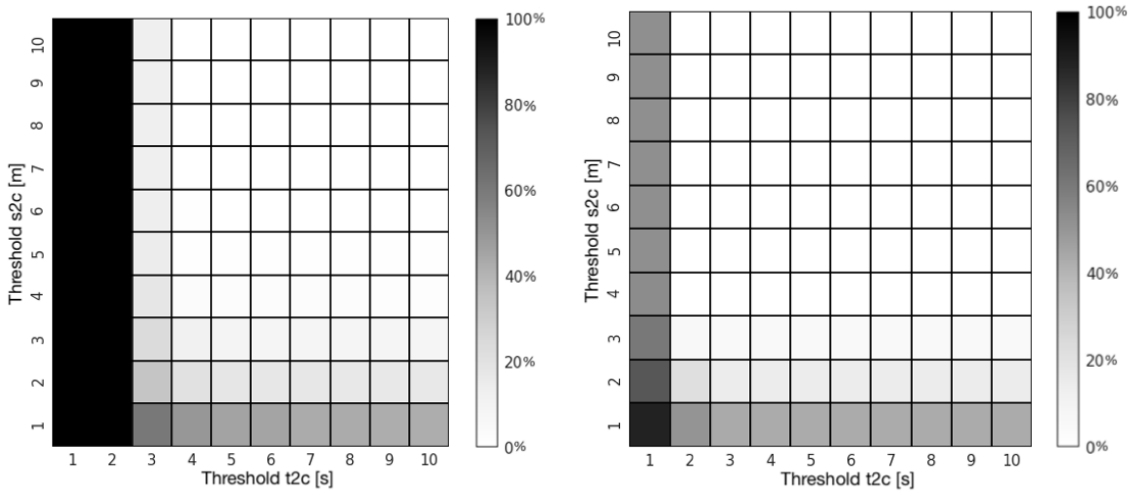
The thresholds $t2c_t$ and $s2c_t$ are the two parameters that mostly affect the running of the algorithm. For this reason, in order to optimize the performance of the CA service, they shall be properly set. Before delving into the analysis, we remind the reader what $t2c_t$ and $s2c_t$ represent:

- $t2c_t$: it is an upper bound on the *time to collision* metric, i.e., the time gap needed for two entities to reach their mutual minimum distance.
- $s2c_t$: it is the upper bound to the distance at which two entities are at the *time to collision*.

Both thresholds depend on several factors, such as the maximum velocity at which vehicles travel, the kind of colliding entity (e.g., vehicle, pedestrian), the human reaction time, and so on. The higher their value, the more likely it is that a pair of entities is considered in collision course. This means that high values allow the algorithm to correctly detect all the collisions, but, at the same time, to generate too many DENM messages, even when not needed, i.e., in situations of low or no danger. On the contrary, low values ensure a few number of unnecessary DENMs, but a percentage of the collisions goes undetected or detected too late. Both cases are potentially dangerous for drivers.

In light of this, we undertook a study on the number of undetected or late-detected collisions, as functions of $s2c_t$ and $t2c_t$. Our aim was to find the minimum values that ensure 100% of collisions correctly detected: higher values only increase the number of false positives, lower values do not permit to detect all the collisions. The results of the study are reported in Figure 3.6.

Looking at the heatmap in Figure 3.6a, it can be noticed that, in the scenario under study, any value of $t2c_t$ equal or lower than 3 s, does not make the system reliable. This means that, due to the delays introduced by diverse factors (e.g., human reaction, braking time), the minimum $t2c_t$ value is 4 s. For $s2c_t$, instead, with a $t2c_t$ equal to 4 s, any value greater than 3 m ensures maximum efficiency to the system, allowing the algorithm to detect in time all the collisions occurred.



(a) Vehicle-with-vehicle

(b) Vehicle-with-pedestrian.

Figure 3.6: Percentage of undetected or late-detected collisions.

As far as the case vehicle-with pedestrian collisions is concerned, Figure 3.6b shows that the minimum $t2c_t$ and $s2c_t$ that guarantee maximum reliability are respectively 2 s and 3 m. The two threshold values are lower because, compared to a vehicle, a pedestrian can stop much faster, almost instantaneously, due to her low speed.

In conclusion, it is worthy to emphasize that these two pairs of values are optimal for our reference scenario, in which vehicles and pedestrians move at a certain speed, drivers have a certain reaction time, and so on. Changing any of these factors, the optimal values of the thresholds may vary.

3.4.2 Performance evaluation of the CA algorithm

We ran two sets of 300s-long simulations, one with the server at the Metro node (i.e., MEC fashion) and the other with the server placed in the cloud. As exemplary values reflecting real-world mobile operators topology, we chose 5 ms and 20 ms for the MEC and cloud case network delays.

In order to get a substantial number of collisions in each simulation, we had to tweak the SUMO's parameters, in order to have always-green traffic lights at the crossings. Furthermore, we did not account for rear-end collisions since we are mainly interested in testing our system in accidents occurring at the intersections.

Table 3.1 summarizes the key parameters used by the CA algorithm and their value.

Table 3.1: CA algorithm parameters.

Parameter	**Value**	
	Vehicle	**Pedestrian**
$t2c_t$	$4s$	$2s$
$s2c_t$	$4m$	$3m$
Max CAM age	$0.8s$	$4s$
CAM frequency	$10Hz$	$1Hz$

Collisions detected

In this section we focus on the effectiveness of the service, by analyzing the number of accidents that can be prevented. In SUMO, a collision is reported every time the polygon describing an entity overlaps with the polygon describing another entity. The results derived from the simulations are reported in Figure 3.7.

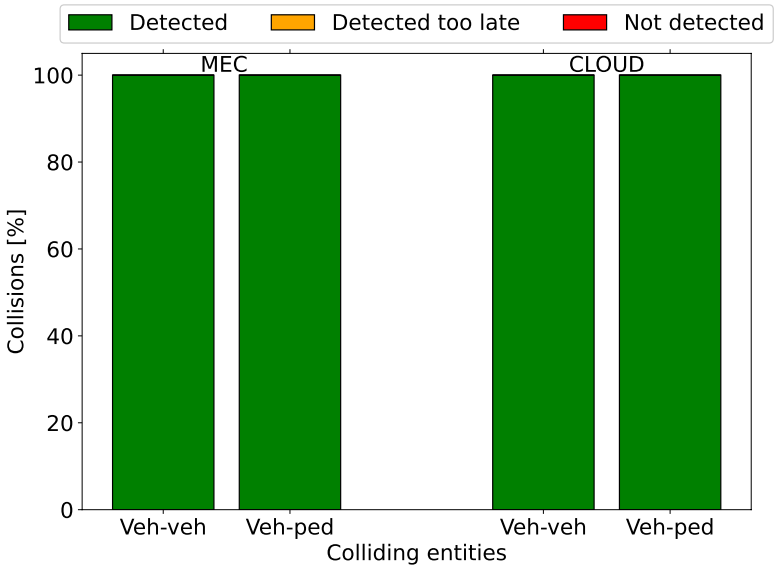


Figure 3.7: Percentage of vehicle-with-vehicle and vehicle-with-pedestrian detected and undetected collisions: MEC vs. cloud.

For what concerns the vehicle-with-vehicle collisions, the results obtained are excellent. On average, we had 17.5 accidents per simulation and all of them, regardless the CA server position, were correctly detected. Moreover, each collision was reported to the driver *sufficiently* in advance, giving him time to react and stop the vehicle.

Next, we can focus on the vehicle-with-pedestrian collisions. The pedestrian generation rate λ_p is relatively low (on average we have 1 pedestrian every 10 s), so the

number of collisions observed is lower than in the previous case. In a first moment, we noticed a little decrease in the effectiveness of our algorithm, with a few collisions undetected. This was observed both in the MAC and in the cloud case. Since such collisions are very few, we scrutinized each of them. It turned out that all these false negatives are due to SUMO, in particular because of the mobility model used for pedestrians at zebra crossings while vehicles are approaching.. Let us consider a car and a pedestrian approaching a free zebra crossing. Since no pedestrian occupies the crosswalk, the car can proceed at its maximum speed. Once the pedestrian enters the zebra crossing, the vehicle sees the obstacle and, according to the SUMO mobility model, starts to slow down attempting to stop. Our algorithm, which is aware of the speed and acceleration of the vehicle, predicts that the latter will never hit the pedestrian and, also, when it stops completely, the pedestrian will be behind the car. As a result, no DENM messages are generated. However, in SUMO, if the vehicle stops on the zebra crossing, the pedestrian, completing the crosswalk, rather than dodging it, physically walks over it. Therefore, in the SUMO logs, this is reported as a vehicle-with-pedestrian collision. By taking into account this issue and not considering such collisions, the service reliability is maximized, since each collision is detected in time.

The results obtained from the simulations are consistent with the performance study presented in the previous section. The values with which we set the thresholds $t2c_t$ and $s2c_t$ allow the algorithm to detect in time each collision occurred in the SUMO simulator.

False positives

In this section we investigate on the quality of DENM messages that are received by the vehicles, in order to find the fraction of false positives, i.e., the DENMs referring to situations of low or no danger. False positives can be harmful and may annoy the drivers, increasing the likelihood they will not react appropriately to future warnings. For this analysis, we studied the following metrics:

- *DENMs sent*: total number of alerts sent by the server to vehicles to warn them about detected collisions.
- *True positives*: transmitted DENMs referring to collisions that actually occurred. They include:
  - *True and timely positives*: DENMs for which the driver had enough time to brake before the collision happen.
  - *True but late positives*: DENMs for which the driver did not have enough time to brake and avoid the impact.
- *False positives*: transmitted DENMs referring to collisions that would not take place.

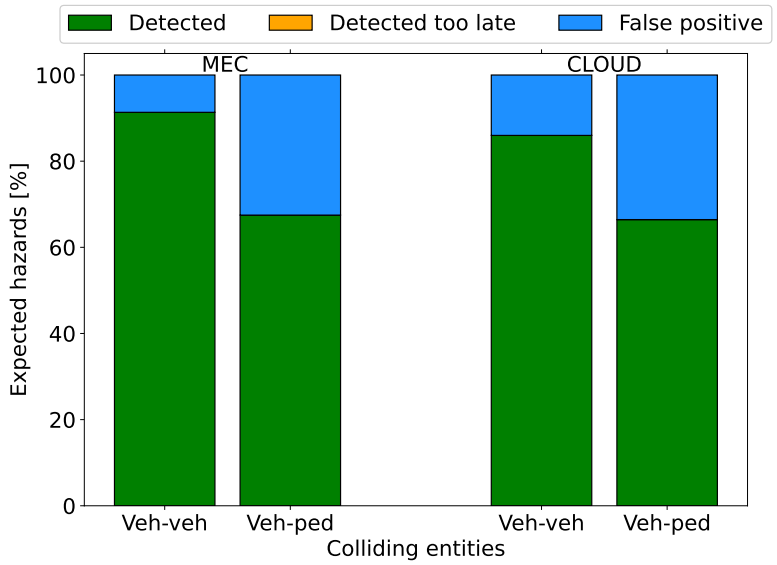


Figure 3.8: Percentage of vehicle-with-vehicle and vehicle-with-pedestrian false positives: MEC vs. cloud.

Figure 3.8 shows the results of this analysis. As regards the vehicle-with-vehicle collisions, the false positives are always lower than 14%. With the MEC-based implementation, this value is 8.66%, smaller than 13.98%, obtained in the other case study. This is due to the lower network latency when the CA server is hosted in a MEC platform. Smaller delays allow more precise computations by the server, thus less unnecessary DENMs transmission.

As far as the vehicle-with-pedestrian false positive alerts are concerned, we can see an increase in cases. Indeed, the rate is around 33% for both server positions. Again, better results are achieved with the MEC-based solution: 32.52% against 33.56%. However, with respect the previous case, this difference is less evident. This is due to the speed at which entities move: pedestrians move slower, thus the higher positioning error made by the server running in the cloud is less incisive.

The higher vehicle-with-pedestrian false positives rate can be explained by considering the characteristics of the pedestrian mobility with respect to cars. Zebra crossings are occupied for a longer time by pedestrians and, since they move at a maximum speed of 2 m/s and the two lanes are 6 m wide each, they will occupy the crossing for about 6 seconds. During this time, it is likely that other vehicles will approach the crosswalk and, if they stop or pass close to a pedestrian, the CA algorithm will generate a DENM, even if no collision actually occurs. By reducing the space threshold $s2c_t$, we could drastically reduce the false positives but then we would get undetected collisions.

3.5 Final remarks

The main purpose of the work described in this chapter was the design and performance assessment of a CA algorithm. We developed a trajectory-based CA algorithm, capable of detecting collisions between vehicles, as well as between vehicles and VRUs. We evaluated its reliability and effectiveness through simulations, by developing, within the SimuLTE-Veins simulator, a C-V2I CA service. The service aims at avoiding collisions between vehicles at intersections and between cars and pedestrians at crosswalks. The performances exhibited by the algorithm were excellent, as all the simulated dangerous situations, involving a possible accident, were correctly detected and drivers were alerted in time.

Chapter 4

Implementation of a MEC-based Collision Avoidance Service in an Experimental Testbed

The 5GT project has aimed at designing and implementing a whole end-to-end architecture capable of offering fine grained and tailored services for a variety of vertical domains with various needs and requirements. In order to assess the performance and the reliability of the 5GT platform, five different vertical industries have been selected to develop each a specific UC, which has been implemented within the platform. One of the UCs was the CA service.

As described in the previous chapter, our first step has been the design of the CA algorithm and its performance assessment through simulations. Here, we present the second step of the work, i.e., the implementation of the CA service within the 5GT-MTP, in our case represented by a MEC platform.

In this chapter we first provide some details on the MEC architecture, and then we describe each module composing the CA service, together with its implementation within the MEC platform. Finally we discuss the performances of our MEC-based service and compare it against a cloud-based solution, i.e., with the CA server hosted in the cloud.

4.1 The MEC architecture

Edge computing comes with the promise of enabling low-latency services, exploiting distributed heterogeneous computing and network resources close to the final user, and reducing core network load by offloading traffic to edge service instances. Recent efforts have brought to detailed standardized architectures for MEC. Figure 4.1 depicts the MEC architecture provided by the ETSI MEC Industry Specification Group (ISG) [67]. The main entities of the MEC architecture are as follows:

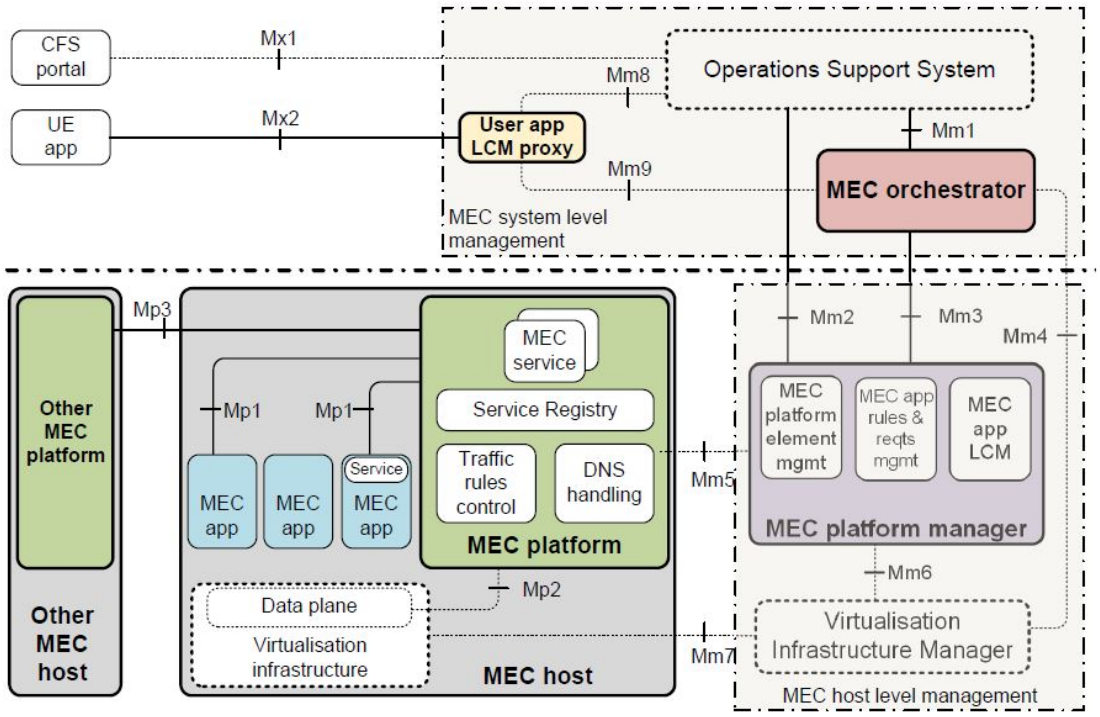


Figure 4.1: Standard MEC architecture [51].

- **MEC host**. It provides an execution environment on which (virtualized) Mobile Edge (ME) applications can run. It is composed of the MEC Platform (MEP) and a virtualization infrastructure (where ME applications are deployed).
- **MEP**. This component represents the interface between the mobile network and ME applications. The MEP leverages the (non-standardized) Mp2 interface to interact with the mobile network and access the user data plane. It exposes MEC services via the Mp1 reference point. The component responsible for MEP configuration and ME application lifecycle management is the MEP Manager (MEPM), which is under the control of the Mobile Edge Orchestrator (MEO).
- **MEC services**. Services discovered and consumed by MEC applications over the Mp1 reference point. The ETSI MEC standards specify a set of MEC services that are provided natively by the MEC platform, such as the Radio Network Information Service (RNIS) [37], local DNS, traffic rules control. The Mp1 interface is also used by third-party MEC applications to register and provide their own services. Note that, it is not necessary that MEC applications provide or consume MEC services.
- **Mobile Edge Orchestrator**: This component has a global view of the whole mobile edge network and is responsible for managing ME applications. The MEO is

the interface between the BSS/OSS and the MEP and host. By interacting with the MEP and the virtual infrastructure, the MEO supports the lifecycle management (e.g., instantiation and termination) of ME applications.

4.2 Testbed implementation

In this section, we illustrate the implementation details of our testbed. It can be divided in four main blocks:

- The MEC-enabled Evolved Packet-Core (EPC) network;
- The procedures for service on-boarding and instantiation within the MEC platform;
- A piece of code to generate and transmit CAMs;
- The CA VNF and the *Cooperative Information Manager* (CIM) VNF, the two main virtual functions composing the MEC service.

Figure 4.2 provides an overview of the interactions among these building blocks. In the testbed, two realistic OAI-based implementations of cellular user equipments (UEs) act as vehicles. Each UE periodically sends messages (CAMs) containing information related to the position, speed, acceleration, and heading of several emulated vehicles towards a third-party database, the CIM[1]. In turn, the MEC-enabled EPC identifies these messages directed to the CIM VNF and applies *traffic redirection* rules to keep them at the edge. The CA VNF, which runs the trajectory-based algorithm presented in Algorithm 1, periodically retrieves the latest vehicle information received by the CIM. When a possible hazard is detected, the CA VNF sends DENM messages towards the vehicles, exploiting again the same traffic redirection rules that the MEC-enabled EPC applied for the uplink traffic.

4.2.1 A MEC platform based on OAI

Our system builds on OAI [79], an open-source implementation of a full LTE network, which includes the RAN and the EPC. On top of this, the MEC platform is implemented. It exposes REST-based API endpoints to the MEO and ME applications, so that they can discover, register, and consume MEC services, such as traffic redirection and, in our case, the automotive service.

In order to implement the Mp2 interface, the extensions to the OAI RAN and the core network elements have been provided. Core network extensions are critical for

[1]Observe that, in the previous chapter, the CIM was not considered as a third-party service.

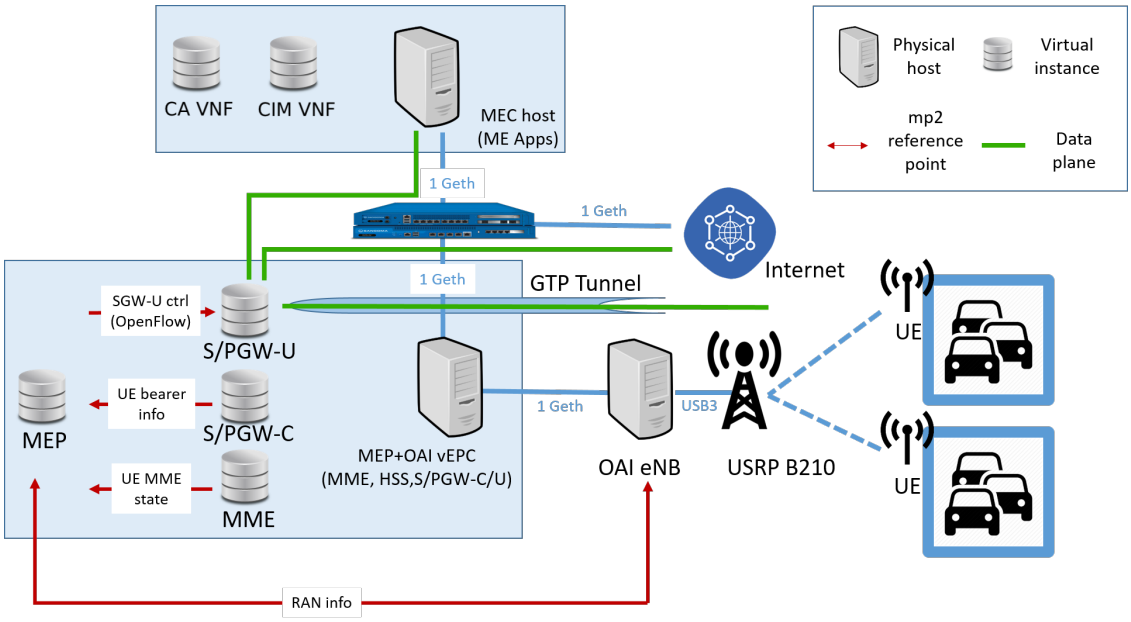


Figure 4.2: Overview on the interaction among the testbed building blocks.

traffic offloading to ME application instances, whereas specific support is needed at the RAN level for retrieving radio network information from eNBs (e.g., per-UE channel quality indications (CQI)), and exposing them to subscribing ME applications.

The Mp2 interface towards the RAN is based on the FlexRAN protocol [43], which is integrated into the standard OAI software distribution. For traffic management, the Control and User Plane Separation (CUPS) paradigm introduced by the 3GPP [89] has been used. According to CUPS, the data- and control-plane functions are separated at the Service/Packet-Gateway (S/P-GW) level. As a result, the S/P-GW has been split into two entities: S/P-GW-C and S/P-GW-U (C for control plane; U for user plane). The former is in charge of managing the signaling, in order to establish the user data plane; the latter is responsible of forwarding the user plane data. In our implementation, the S/P-GW-U leverages a version of OpenVSwitch (OVS), patched to support GTP packet matching. When requested over its Mp1 interface, the MEP, through OpenFlow commands, installs traffic redirection rules on the S/P-GW-U to offload data to the MEC applications. The MEP needs to be aware of specific UE bearer information (i.e., UE IMSI, GTP tunnel endpoint identifiers, UE and eNB IP addresses) in order to appropriately install the rules. This information is available at the S/P-GW-C level thus, it was needed to modify the OAI EPC code to share it to the MEP via its REST Mp2 interface.

In our testbed, ME applications are run on the MEC host as VMs, directly on top of the *kvm* [55] hypervisor. However, this MEC platform is also compatible with Virtual Infrastructure Managers (VIMs) such as OpenStack [81], while, as regards container-based solutions, only lxd [59] is supported.

Figure 4.2 shows our MEC testbed setup with their main interfaces and components.

The OAI EPC is virtualized, thus the HSS, MME, and S/P-GW run as separate VMs on a single physical machine. The same machine also hosts the MEP. The OAI eNB instead, due to its real-time constraints, runs on a dedicated machine, to which a USRP B210 RF board is attached.

4.2.2 Vertical service components as MEC applications

In this section, we present the procedures for the deployment of our automotive service components as MEC applications on our platform, i.e., preparation, instantiation and service discovery.

Application preparation and on-boarding - The ETSI MEC standards require that MEC applications are characterized by an application descriptor (AppD) [36], which is prepared by the service provider (in our case, the automotive vertical) as part of an application package. It provides a lot of pivotal information for the application deployment, such as:

- Reference (URL) to the actual application image;
- Application latency requirements;
- Minimum resources requirements (e.g., amount of computing resources that should be allocated for an application instance)
- MEC services exploited or consumed by the application;
- DNS rules and traffic filters.

The latter two define the characteristics of the traffic that should be forwarded toward the MEC application instance (e.g., traffic flows matching a specific protocol-destination and address-port tuple).

The vertical industry provides an application package for on-boarding to the OSS/BSS via the Customer Facing Service (CFS) Portal. Then, the OSS/BSS on-boards such an application package to the MEC system by communicating with the MEO over the standardized Mm1 reference point, thus making it available for instantiation.

Our automotive service was composed of two main application packages, namely, the CIM VNF and the CA VNF. Our service design did not preclude monolithic implementations, where all components are *(i)* provided by the same entity, *(ii)* developed in a single package by the vertical, *(iii)* deployed in a single, standalone, MEC application instance, but it would have limited deployment flexibility and scaling capabilities. Furthermore, we expect that in a real-world implementation the CIM is provided by a different entity, such as a transportation authority, and would expose its information to multiple MEC applications (such as our CA VNF). For this reason, we opted for a microservices based implementation, where each component is on-boarded and instantiated

separately. As discussed later, appropriate, standards-based service registration and discovery procedures are used so that the application components (including those aboard the vehicles) can discover and share information with each other.

In conclusion, our vertical service design included the following two main MEC application components:

- **CIM VNF**, which included both the CAM Receiver module and the Information Manager module. The CIM was characterized by a modular design, with all its components operating as separate, networked sub-services.
- **CA VNF**. It was composed of the CA Manager module, the CA algorithm, and the DENM-Generator (DENM-G) module. The CA algorithm communicated with both, as it received the CAMs to be processed by the CA Manager and, in case of detected collision, forwarded the information needed to generate the DENM messages to the DENM-G.

Further details on these VNFs are provided in Section 4.2.4.

Instantiation - Once the application packages have been on-boarded (as per the request of the vertical service provider over the CFS portal), the OSS/BSS uses the *Application Lifecycle Management Interface* of the MEO (Mm1 reference point) to instantiate the CIM and the CA VNFs. Each request includes the identifier of the respective application package, according to the procedure detailed in ETSI MEC 010-2 [36]. The information included in the AppDs is used to: *(i)* configure the MEP for traffic redirection, *(ii)* update the the MEC service's DNS, and *(iii)* register the necessary service API endpoints of each component with the MEP.

Service discovery - As regards the CIM, the following requirements need to be met:

- Since CIM virtual instances are created dynamically within the MEC system, and each client subscribed to the CA service needs to transmit its CAMs to the CIM instance covering its region, a mechanism to steer CAMs to the appropriate MEC instance has to be put in place. This needs to take place transparently and with minimal UE involvement.
- In a real-world implementation, the CIM service may be provided by a third-party, e.g., a transportation authority, and a single CIM instance may have to provide its information to multiple CA instances. Consequently, upon the deployment of a CIM MEC instance, its service endpoint needs to be registered with the MEP, so that it can be discovered by CA applications.

In order to allow vehicles to discover the IP address of the CIM covering their area, we used a combination of standard DNS mechanisms and MEC capabilities. We reasonably assumed that OBUs on vehicles are pre-programmed to search for the CIM at

a well-known DNS name. Thus, to encode it, we used the `appDNSRule` field of the AppD. Once the CIM VNF is instantiated, the MEO instructs the MEP(M) via the Mm3 reference point to update the MEC DNS database. A new entry is then created to resolve the CIM name to the IP address of the new MEC application instance[2].

If the vehicular UEs are instead pre-programmed to communicate with a fixed CIM IP address (or receive this IP address by a centralized control entity), it is needed to add specific `appTrafficRule` entries in the AppD, so that appropriate traffic redirection rules can be set up in the MEC platform. In particular, this field enables the use of traffic filters that can match specific flows, identified among others by tuples composed of service IP address, port, protocol. Upon service instantiation, the MEO extracts `appTrafficRules` from the AppD and, by communicating with the MEPM via the Mm3 interface, apply them. The MEPM, in turn, gets the MEP's traffic rules service (in our implementation, using a REST interface), and the latter eventually applies them to the S/P-GW-U over the Mp2 interface. This type of traffic redirection is based on SDN and is transparent to the UE: CAMs are sent to the well-known IP address and port of the CIM, and the S/P-GW-U redirects the traffic to the IP address/port of the MEC instance, by applying packet-rewriting OpenFlow rules installed by the MEP.

CA VNFs, on the other hand, consume the CIM VNF data through the Mp1 interface. Upon the CIM instantiation, the MEO extracts the `appServiceProduced` field from the AppD. This field describes the service endpoint exposed by the CIM that should be used by the CA Manager component, in order to access and consume the input to the CA algorithm. Lastly, the MEO can finally add the service to the MEP Service Registry.

4.2.3 The vehicle simulator

In our testbed, two different UEs acted as vehicles. The mobility traces describing the pattern of such emulated cars were obtained previously, by running the well-known SUMO simulator. We sampled the mobility traces of each car every 100 ms (10 Hz), recording the key information of vehicle movements, such as position, speed, acceleration, and direction. With each obtained sample we created the corresponding CAM, which was transmitted towards the eNB of the OAI cellular network and offloaded to the CIM service.

The radio interface of the two UEs used in the testbed leveraged the standard OAI UE implementation. Each UE was emulated by a computer, equipped with an octa-core processor at 2 GHz and 16 GB RAM, connected to a USRP B210 RF board [38]. The over-the-air communication was improved with the use of a pass-band filter, which reduced undesired interference at the receiver. On both UEs, the software for transmitting CAMs and receiving DENMs (the latter being alert messages sent by the CA VNF

[2]This means that vehicular UEs use the (local) MEC DNS to resolve the domain name of the CIM. However, this is a reasonable assumption because the UE DNS server is assigned by the mobile network and MEC operator.

to the vehicles) was installed. This software has been called *VehicleSimulator* and it is a C++ standalone Linux application. An overview of the VehicleSimulator architecture and its main components is depicted in Figure 4.3.

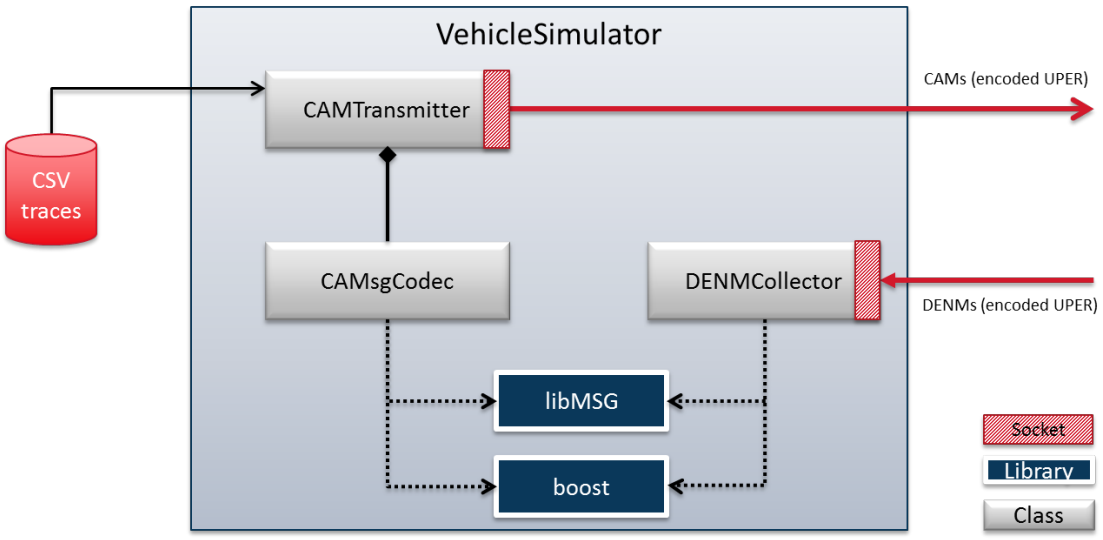


Figure 4.3: VehicleSimulator architecture.

The main class of the VehicleSimulator software is the *CAMTransmitter*, which, as the name suggests, is in charge of transmitting CAMs. In order to perform such an operation, the CAMTransmitter receives as input a CSV file, in which each line corresponds to a CAM. Each CAM is stored in the RAM in order to guarantee a faster access to the information. When it is time[3] to generate a CAM, the CAMTransmitter starts a new thread with which manages its creation. This thread calls the *CAMsgCodec* class, which:

- creates the CAM structure[4], allocating memory to each CAM field;
- parses the CSV line passed by the CAMTransmitter;
- updates the CAM structure with the acquired information;
- checks the consistency of the CAM structure with the standard [33];
- encodes the CAM structure according to the Unaligned Packet Encoding Rule (UPER) to obtain the byte array.

[3]Each row of the CSV file also contains the time at which the CAM should be transmitted.

[4]The CAM structure is provided by the library *libMSG* of the open-source compiler *ASN1* [83].

After that, the CAMTransmitter forwards the CAM via UDP socket towards the OAI-based eNB and stores the time instant at which such a transmission took place. Observe that, since the CAMTransmitter features a multi-thread structure, it is able to encode CAMs at the millisecond time-scale, and therefore, it can transmit messages generated by multiple vehicles.

When a collision between two cars is detected, the CA service has to transmit a DENM towards the involved vehicles, i.e., towards the UEs emulating the two vehicles. To send the DENM to the correct UE, the CA VNF needs to know the UEs' IP address. This information is obtainable from the CIM VNF, which stores also the IP address of each vehicle in the system. At VehicleSimulator side, the DENMs are received (at a specific UDP port) and encoded by the *DENMCollector* class. To guarantee better performance, the latter exploits two different threads. The first listens to the UDP socket used by the CA service and it performs the decoding and storing of the DENMs in a dedicated queue in the RAM. The second thread, instead, processes the information contained in such a queue and records the time spent by the VehicleSimulator to decode each DENM. The socket operations and the management of the queue structure have been optimized by using the boost C++ libraries.

4.2.4 The automotive MEC service

The MEC safety service implemented on the testbed had two core functions: the CIM VNF, in charge of receiving and storing CAMs from the vehicles in the monitored area, and the CA VNF, which ran the CA algorithm presented in Chapter 3 and forwarded DENMs to vehicles involved in a detected collision. In this section, we describe in detail their design and implementation within the MEC testbed.

The CIM VNF

All CAMs originated by the simulated vehicles (i.e., the two UEs) are redirected by the MEC-enabled EPC towards the CIM VNF. The CIM is an evolved version of the Local Dynamic Map (LDM), a system standardized by ETSI that maintains road traffic information that are consumed by various ITS. Data can be received from a wide range of sources, such as vehicles, infrastructure units, personal ITS stations, and so on. The LDM ensures high security to the stored data. For instance, it can provide information on the surrounding vehicles and RSU to any authorized application. The CIM is based on the same concept, it is a data storage located in a MEC host, able to receive all information relevant to the CA service from vehicles moving in a given region. As mentioned, the structure of the CIM can be split into the following two main blocks: the *CAM Receiver* and the *Information Manager* (IM). In our testbed, the CIM was virtualized in a VM with 2 cores at 2 GHz each and 4 GB RAM.

The first block, i.e., the CAM Receiver, is a C++ standalone application, whose structure is represented in Figure 4.4. The main performed operations are the following:

- the reception of the encoded UPER CAM messages from the vehicles;
- the decoding of the CAMs and the extraction of the vehicle data;
- the transmission of the extracted data to the IM.

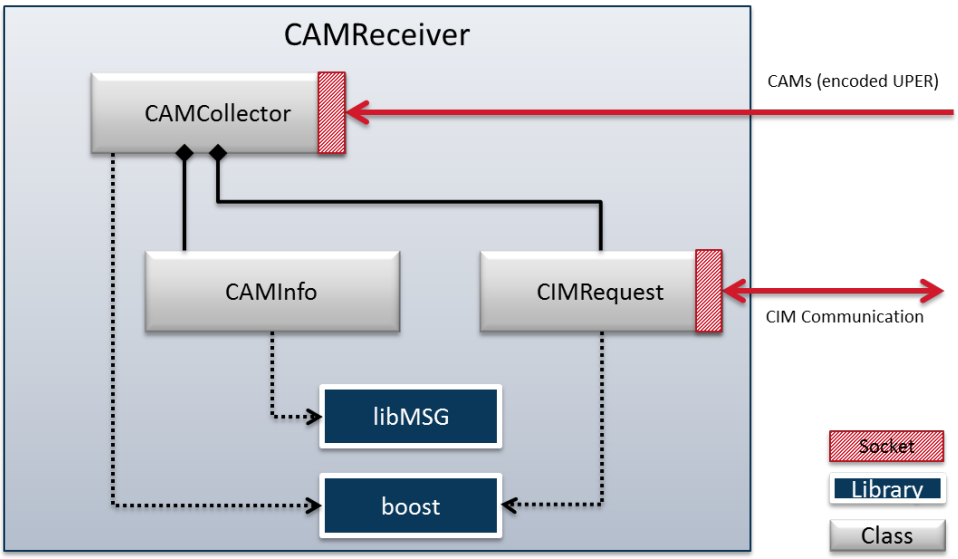


Figure 4.4: CAMReceiver architecture.

The most important class of the CAMReceiver is the *CAM collector* class, which is responsible of receiving, over a UDP socket, the CAM messages from the vehicles. In addition, after having decoded them, it sends the received information to the IM via the *CIM Request* object. The decoding of CAMs is provided by the *CAM Info* class. Upon the reception of each CAM, a new thread is started: a specific message is created and prepared for a UDP/TCP connection to the IM and the computation time of the CAM reception operation is recorded.

The second fundamental block of the CIM is the IM, a standalone application implemented as a JAVA 8 runnable JAR file, whose structure is shown in Figure 4.5. The IM does not have a user interface but it is provided by a separated Web Portal based on Tomcat 9. The IM presents a single input port and an arbitrary number of output ports. The input port can manage multi-thread connections and is used to receive the CAMs from the CAM Receiver. The CIM can be configured in order to manage only the CAMs coming from vehicles in a specific area, while all the other are ignored. Such an area is a circle, and it can be defined by specifying the latitude, longitude of the center and the radius. Also the Web Portal uses the input port, in order to configure the Information Manager.

The output ports of the CIM can be added or removed from a previously defined list while the CIM is running. Each output port is managed by a module called *CAM*

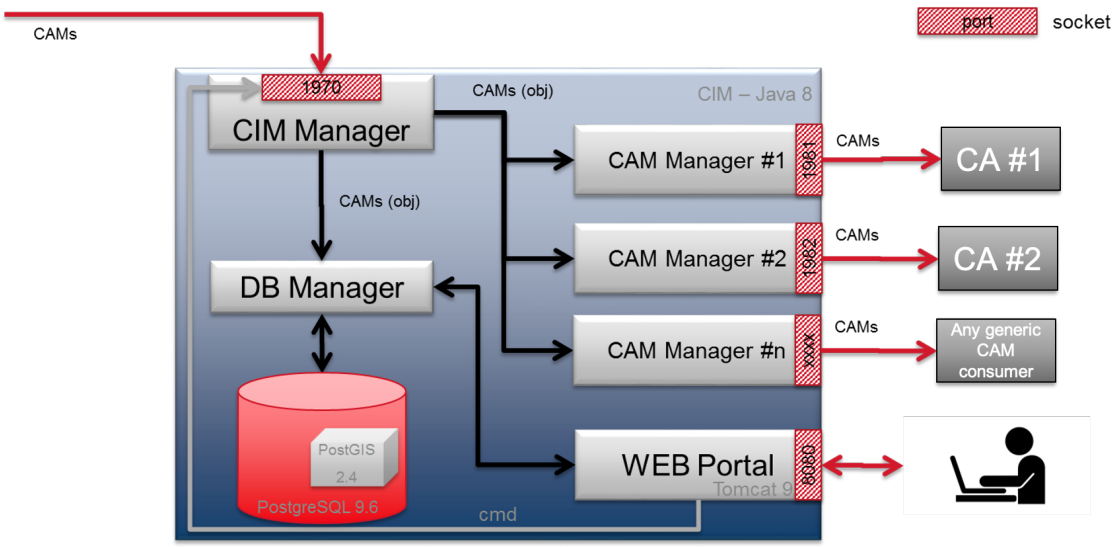


Figure 4.5: Information Manager architecture.

Manager in charge of satisfying the queries of a specific CAM consumer (i.e., a particular CA instance). All CAM Managers receive a copy of the CAMs collected by the IM, from a module called *CIM Manager*. Each CAM manager is configured to manage a specific circular sub-area inside the whole monitored region (i.e. a crossroad under the control of a particular CA instance) and only the CAM messages generated inside such a sub-area are provided to the corresponding CAM consumers (look at Figure 4.6).

The IM can manage two storage mechanisms. The first is a volatile mechanism, used by the CAM Managers, that allows the storage of the received CAM messages in the RAM memory. On the contrary, the second one is a persistent storage mechanism, in which CAMs are stored in a PostgreSQL relational database. Clearly, the volatile mechanism guarantees a faster access to the acquired CAMs. As can be easily guessed, the amount of memory needed to store CAMs is directly proportional to the number of vehicles traveling in the region monitored by the CIM. Therefore, the volatile mechanism satisfies the requirement of readiness, which is of utmost importance for a CA algorithm, but it is not suitable for other needs, such as the statistical analysis on the messages collected by the CIM. Through the use of a PostGIS extension, the persistent storage mechanism can support statistical analysis through searches of messages inside geographical regions. With these results, for example, we can improve either the CA algorithm or the definition of the area controlled by the algorithm, i.e., the area under the CAM Manager control. In order to avoid a worsening of the IM performance owing to the high workload of the persistent storage, the *DB Manager* module implements a multi-thread storage mechanism. In particular, a queue for storage sessions is implemented and served when it is more convenient. This approach has the effect that persistent storage mechanism is performed with delay and with the requirement

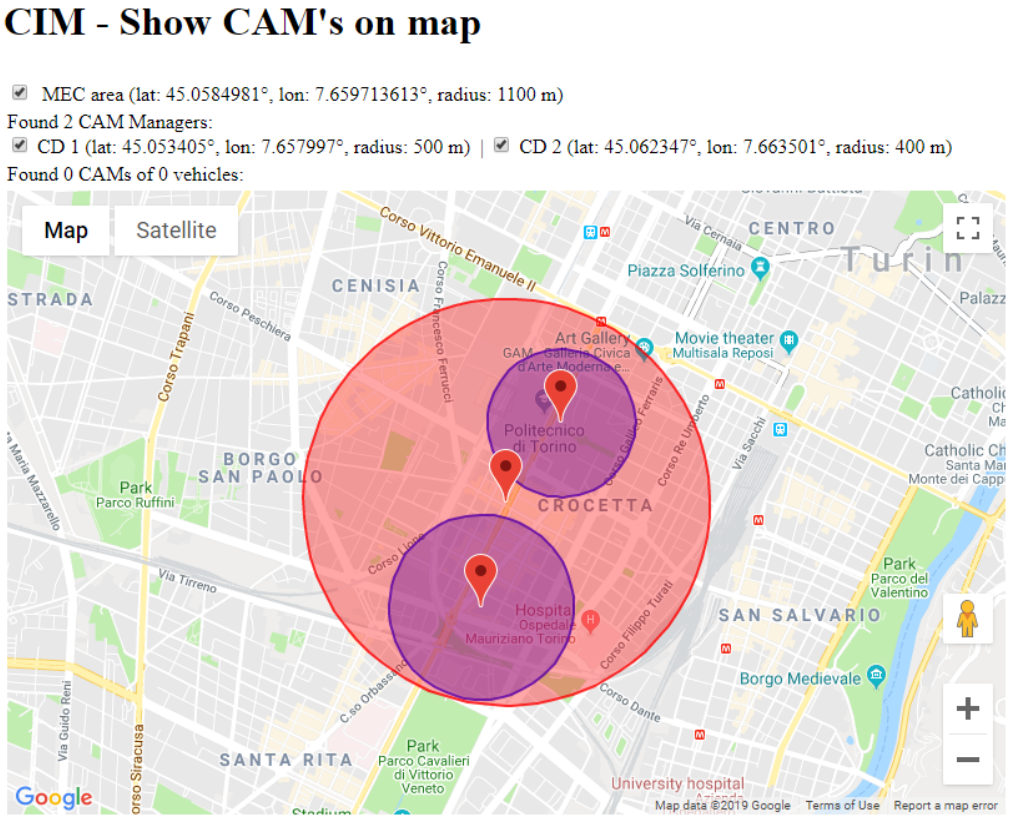


Figure 4.6: Screenshot of the CIM Web Portal: example of the map used for selecting the area monitored by the CAM manager.

of additional available memory.

The CA VNF

The scope of the CA VNF is to detect imminent collisions between vehicles on a specific portion of the scenario controlled by the CIM. In our design, the CA VNF has been developed as a standalone C++ application. It was deployed in the MEC host on a dedicated VM with 1 core at 2 GHz and 2 GB RAM. An overview of the CA VNF is shown in Figure 4.7.

One of the main components of the CA VNF is the *CA Manager*. The CA Manager can query a set of, not necessarily all, CAM Managers at the CIM VNF, according to the monitored area. This implementation choice was motivated by the assumption that the CIM belongs to a third-party that does not run, but only responds to the CA VNF.

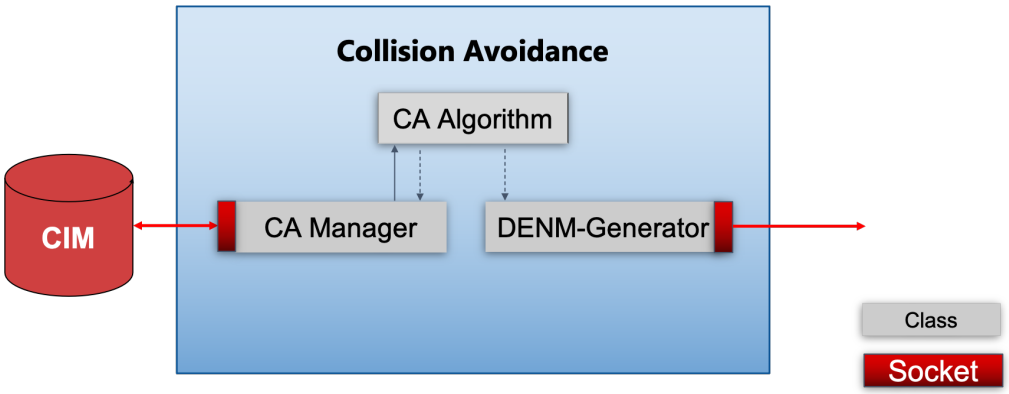


Figure 4.7: CA VNF architecture. The CA Algorithm is the one presented in Chapter 3 and described in Algorithm 1.

Every 5 ms[5], the CA Manager queries the latest CAMs to the CIM over a TCP connection. The 5ms-threshold that we set represents an optimal trade-off between the additional delay due to sequential queries to the CIM and the computational load of the CA algorithm. The CIM provides CAMs in an aggregate form in JSON format to the CA Manager. In turn, the CA Manager interprets the response and passes the new CAMs, if any, to the CA algorithm. The latter updates the trajectories of the vehicles whose CAMs have just been received, and compares them with the trajectories of all known vehicles traveling in the monitored area. As described in Section 3.3.3, in order to improve the system performance we introduced two enhancements in our algorithm: *(i)* the *range of action* filter, to avoid the comparison of trajectories of vehicles far from each other; *(ii)* the cancellation from memory of the information concerning all those vehicles whose last CAM received is older than 0.8 s. Observe that, since the CA VNF queries the CIM at a constant rate, the memory cleanliness from stale information happens only if the collision detection phase lasted less than the query cycle duration, i.e., 5 ms.

Each time a pair of vehicles is detected on collision course, the CA algorithm creates a JSON with the relevant information and passes it to the DENM-G module. The latter, after the reception of such JSON data, prepares and transmits unicast DENM messages to the involved vehicles. In order to avoid an excessive number of duplicated alerts, we imposed to the DENM-G not to generate the same DENM message for a given collision[6] more than twice every 100 ms. Once these messages reach the two candidate vehicles, the HMI processes them and, through a sound for example, warns the drivers about the imminent danger.

[5]It is possible to set a different value, according to the needs.

[6]A collision is identified by the couple of colliding cars and the location of the accident.

As far as the architecture of the DENM-G is concerned, its two main classes are the *DENManager* and the *DENMTransmitter*. The former parses each JSON received by the CA algorithm and prepares the DENMs (e.g., fills the messages fields); the latter is in charge of transmitting the DENMs via the UDP socket. In order to transmit a DENM message, the DENMTransmitter runs the *DENMsgCodec* class, the equivalent of the CAMsgCodec previously described. Therefore, such a class creates the DENM structure, updates it with the information contained in the JSON received by the CA algorithm, checks the consistency with the standard, and finally encodes the DENM obtaining a byte array ready to be transmitted.

4.2.5 Related work and MEC testbed implementations

An extensive body of works have studied MEC architectures (e.g., [96]), but concrete MEC system implementations are scarce. In [93], Subramanya et al. present the design and implementation of a MEC platform with the goal of requiring no modifications both at the RAN and EPC. Their idea is to maintain the necessary UE context information to carry out traffic steering by intercepting the S1 control plane traffic (S1-C) between the eNB and the MME, during UE attachment and handovers. The work of Schiller et al. proposes a similar approach, in order to avoid any interaction between the EPC and the mobile edge system. However, we argue that currently it is not possible to have full MEC functionality in a way transparent to the network, for two different reasons. Firstly, if S1-C traffic is encrypted, this approach does not directly work, since it is not possible to intercept the S1-C messages to monitor the necessary UE state at the MEC platform level; secondly, one of the main MEC platform services is the RNIS: without having an interface to the RAN (part of the Mp2 reference point), it is not possible to retrieve real-time radio network information from the eNBs. Therefore, we opted for a solution that requires a set of necessary extensions at the EPC and eNB levels in order to implement the Mp2 interface, which tailors our solution to OAI, particularly regarding the RAN part.

Our implementation bears more similarities with LL-MEC described in [74], a MEC design also focused on OAI. As in our case, LL-MEC uses SDN techniques for control-user plane separation, as well as the same southbound protocol [43] for retrieving RAN-level information from OAI eNBs. Other implementation differences aside (e.g., different Mp2 interface towards the EPC control/user planes), our MEC system further includes a standards-compliant implementation of the Mm1 reference point of the MEO towards the OSS/BSS, an RNIS interface that fully complies with ETSI MEC 012 [37], and platform components for MEC service discovery and registration.

4.3 Performance metrics

In this section, we present the different metrics we used in our experiments to assess the performance of the MEC testbed and of the MEC CA service presented above. Specifically, we present latency related metrics and metrics related to the ability of our road safety service to detect collisions. Using such metrics, we then compare our MEC-based solution against a cloud-based implementation of the CA service, i.e., without exploiting the MEC traffic redirection rules and with the CA and CIM VNFs running on a cloud data-center.

4.3.1 End-to-end delay and application processing times

To evaluate the ability of the CA service to alert a vehicle on a collision course with real-time information, we used the end-to-end delay metric. The end-to-end delay is computed only with CAMs that trigger a DENM, and it is defined as the time that elapses between the transmission of a CAM by a vehicle and the reception, by the same vehicle, of the DENM that such CAM triggered. Figure 4.8 depicts all the components constitut-

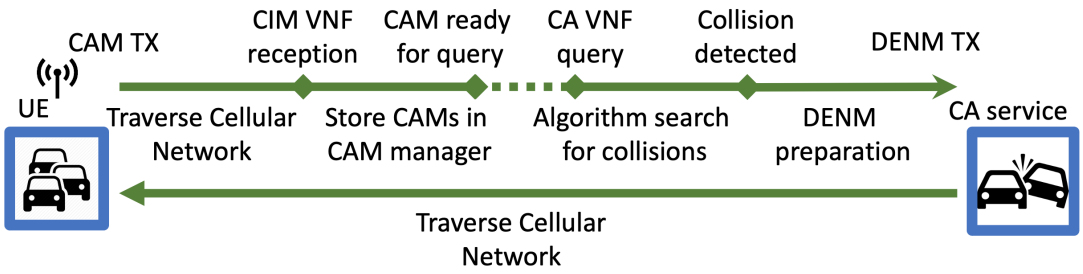


Figure 4.8: Components of the end-to-end delay.

ing this metric. It can be observed that the end-to-end delay comprises both network delays and processing times. Specifically, as network delays, we considered the time needed by a CAM to be delivered to the CIM and, on the return path, i.e., the time needed by the DENM message to reach the vehicle that previously sent the CAM. As processing time instead, we considered both the one at the CIM VNF and the one at the CA VNF. As regards the CIM processing time, we took into account both the time needed to decode CAMs and to present them at the CAM managers for the CA queries. For the CA VNF instead, we considered the time needed by the algorithm to search for collisions and the time to generate and transmit the DENM message to the corresponding vehicles.

While network latency strongly depends on whether the CA/CIM VNFs run in the cloud or in the MEC, their processing time depends only on the computational resources

assigned to the VMs running them and on the incoming traffic that they process. Therefore, in order to profile the processing times of the two VNFs composing the automotive service, we also performed tests varying the number of vehicles in the analyzed scenarios.

4.3.2 CA service performance

To evaluate the performance of the CA service when implemented in our experimental testbed, we first built a ground truth for the collisions. We prepared diverse mobility traces and used each of them to run simulations. For each simulation we got a SUMO error-log file reporting all the occurred collisions between vehicles if no CA service is implemented. Since the same mobility traces are also used in our testbed, analyzing the DENMs correctly received by the two UEs, we easily computed two fundamental performance metrics: *(i)* the percentage of detected collisions, and *(ii)* the percentage of false positives.

For each detected collision, according to the methodology illustrated in Section 3.3.4, we assessed whether it has been detected in time or too late. Moreover, as regards the false positive alerts, i.e., an alarm is raised but no collision occurs, we computed the minimum distance between the trajectories of the two involved vehicles. In such a way, it was possible to understand how far from an actual collision the involved vehicles were.

4.4 Performance evaluation

In this section, we first present the scenario considered for our performance evaluation. Then, we discuss the obtained results in terms of end-to-end and processing delays, and collision detection performance of our CA service in its MEC and cloud implementation.

4.4.1 Reference scenario

The reference scenario is reported in Figure 4.9 and it is similar to the one described in Section 3.3.2. The main difference is the absence of pedestrian areas since in this case we focused entirely on collisions between vehicles at intersections. Vehicles, which were emulated by two UEs, traversed the scenario from north to south (or vice-versa), and from east to west (or vice-versa). In order to simplify the DENM transmissions towards the pair of vehicles involved in a collision, we used one of the two UEs to emulate only vehicles in the north-south direction, while the second UE to emulate the presence of the vehicles traveling in the east-west direction. Finally, to evaluate how the number of cars in the system affects the service performance, we used three different values of vehicle density: *(i)* high, i.e., 20 vehicles/km, *(ii)* medium, i.e., 14 vehicles/km, and *(iii)* low, i.e., 7 vehicles/km. The arrival times of cars into the system followed an exponential

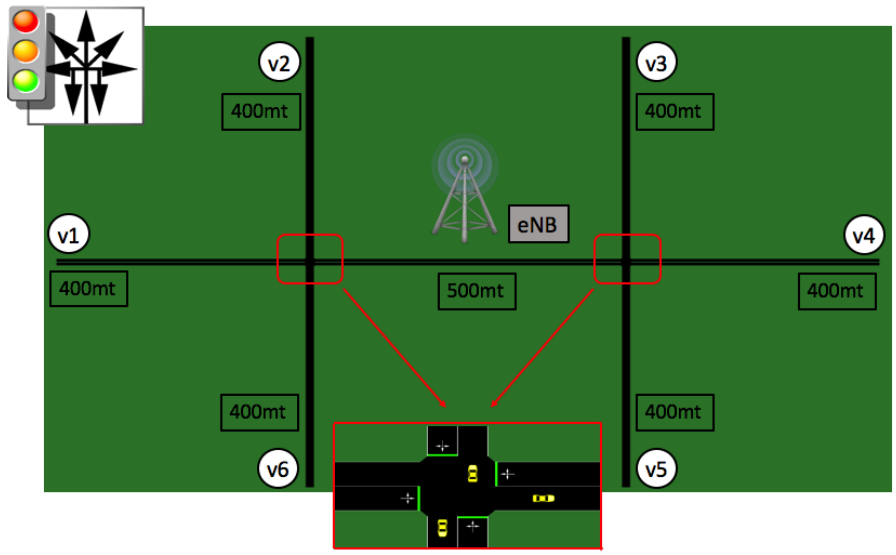


Figure 4.9: Screenshot (from SUMO) of the urban area monitored by the MEC CA service.

distribution, with mean set to the aforementioned values in the three different cases, respectively. For each vehicle density, we performed 10 different runs of 300 seconds each. According to the study presented in the previous chapter in Section 3.4.1, we set the threshold $t2c_t$ and $s2c_t$ (which are key parameters to guarantee high performances to our CA algorithm) to 4 s and 4 m, respectively.

4.4.2 End-to-end and processing delays

As described in Section 4.3.1, the end-to-end delay is composed of three main components:

- The network latency;
- The CIM storage and processing times;
- The time to run the CA algorithm and prepare DENM messages.

Note that, the only difference between our CA MEC implementation and the cloud counterpart is represented by the network latency. In order to account for the additional delay required by the CAMs to reach the CIM when deployed in a cloud server, we ran two measurement campaigns. With our OAI UE we first pinged 10,000 times the CIM in the MEC and collected the measured network latency. Then, to evaluate the impact of traversing a real cellular EPC to reach a cloud server, we pinged with a commercial smartphone the Amazon data-center closest to Turin (where our testbed is located), i.e., the one in Paris [12]. Figure 4.10 shows the cumulative distribution function (CDF) of

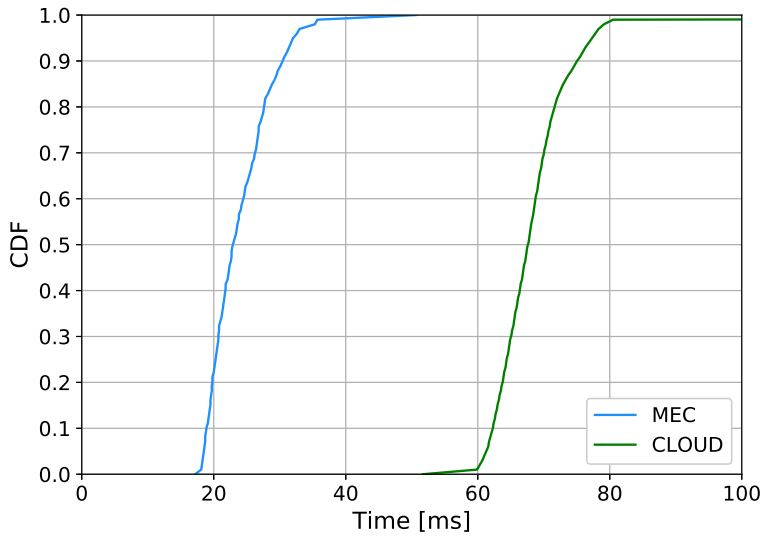


Figure 4.10: CDF of the network delays measured in the MEC and in the cloud experiments.

the obtained network latency in the two cases of MEC and cloud-based implementation.

Interestingly, the delay difference between cloud and MEC approximates a Gaussian distribution, with mean 44.24 ms and standard deviation 8.36 ms. Thanks to this result, we leveraged *NetEm* [73] to mimic the effects of the cloud in our testbed. Specifically, when evaluating the performance of the CA service in the cloud, we introduced a Gaussian distributed additional delay both at the ingress and the egress port of the CIM and CA VNFs, respectively. The distribution of the total delay added to each packet was consistent with the measured difference between MEC and cloud in our measurement campaigns. For what concerns the processing times, which statistically are the same in the cloud and the MEC case studies, we report in Figure 4.11 the CDF of the CA VNF processing time for the three vehicle densities we selected. A higher number of vehicles in the system clearly affects the performance of the CA VNF (mainly, the CA algorithm it runs). Indeed, times increase on average by 50% between the three vehicle densities. This is mainly due to the fact that the number of trajectory comparisons performed by the algorithm increase, and so does the processing time. Nevertheless, even for the highest value of vehicle density, in 99.9% of the cases, the CA VNF processes all CAMs and, when needed, triggers the required alarms, within 5 ms, i.e., before querying again the CIM.

For what concerns the CIM VNF processing times, they are barely affected by the increase of vehicles in the monitored area. Even in the case of high density, the worst-case processing time at the CIM is below 0.5 ms. This is due to the fact that the CIM handles (theoretically) more vehicles than the CA and, to mimic a real deployment, we assigned to the CIM VM twice the capacity (in terms of RAM and CPU) of the CA VM.

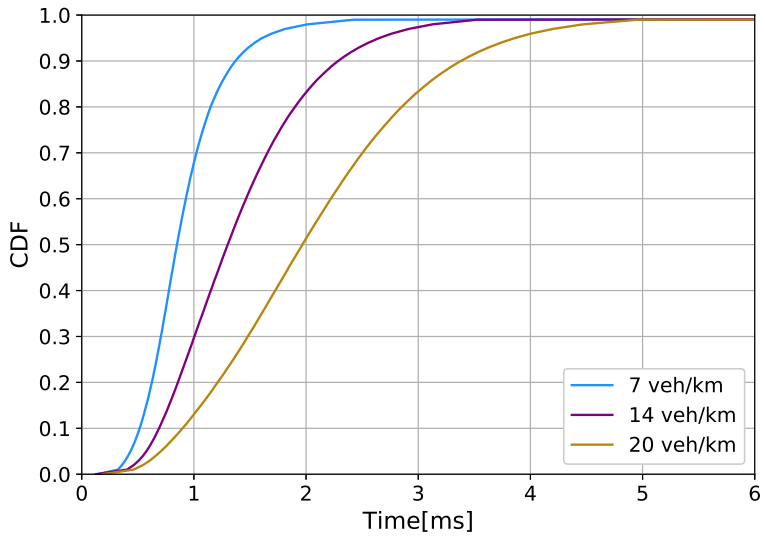
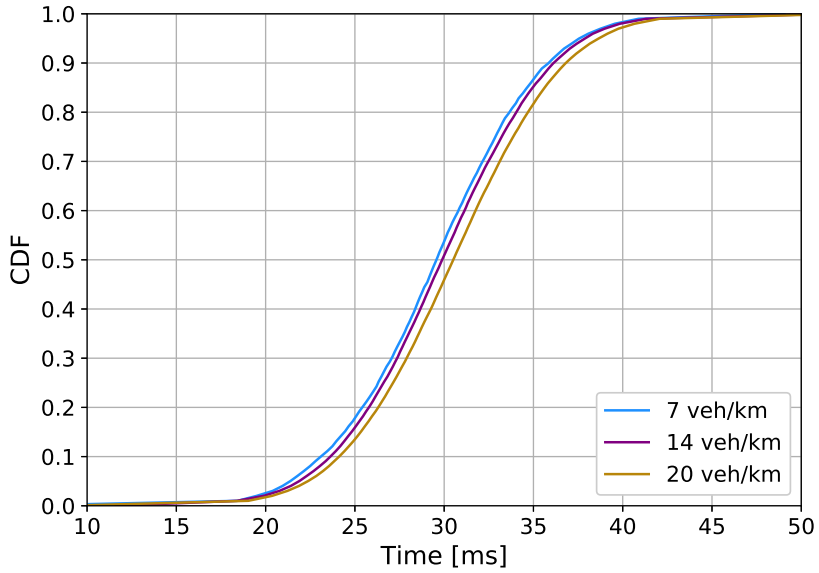


Figure 4.11: CDF of the processing time of the CA VNF.

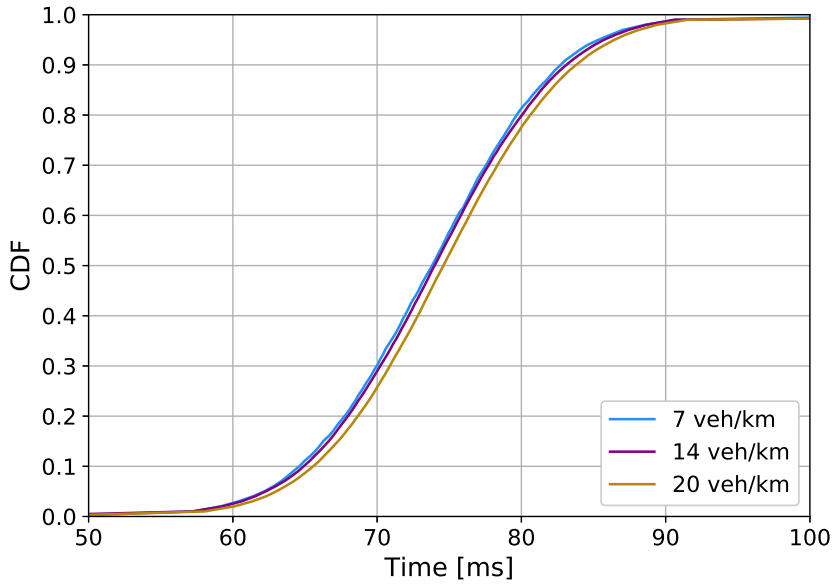
Finally, Fig.12a and Fig.12b depict the experimental CDF of the end-to-end latency of our MEC and cloud-based implementations, respectively. Each curve is obtained considering all the DENMs received by the vehicles in the 10 different runs performed for each vehicle density. In each test, the average number of collisions is 91 for the high-density case, 44.6 for the medium density, and 11.2 for the low density. Given the algorithm used and the parameters setting (i.e., the thresholds $t2c_t$ and $s2c_t$), the maximum number of DENMs that can be generated for a given collision is 80. Indeed, in the best case, two cars on a collision course start receiving DENMs 4 s before the expected impact, once for every CAM they transmit, i.e., two CAMs every 100 ms.

In both end-to-end latency distributions, we can notice a discrepancy between the summation of the three components we presented above and the total latency. Such discrepancy is mainly due to two reasons: *(i)* the CA Manager does not query the CIM as soon as a CAM is logged into the corresponding CAM manager, but only once every 5 ms; *(ii)* each CAM has to flow from a VM to another and such latency is not taken into account in any of the components we presented. On average, the observed discrepancy is 4.34 ms for both implementations, which is in line with the two components we cannot measure.

In our MEC-based implementation, for any vehicle density considered, the 99.9% of the end-to-end latency values we obtained was below 50 ms. In particular, the average end-to-end latency measured was 29.55 ms for the low-density case, 29.89 ms for the medium density, and 30.5 ms for the high density. As regards our cloud-based implementation, instead, the end-to-end latency has never fallen below 50 ms. On average, the latter were 44 ms larger than the end-to-end delays in the MEC-based implementation, which exactly corresponds to network latency differences.



(a) MEC-based implementation.



(b) Cloud-based implementation.

Figure 4.12: CDF of the end-to-end delay as a function of the vehicle density.

In order to evaluate if the end-to-end latency achieved by our implementation is good enough, we can consider as a reference the cycle time of LiDAR sensors on board of a vehicle [30]. These sensors typically update their information every 60 ms so data

carried by a DENM are consistent with on-board sensors, only if the maximum end-to-end latency does not exceed that value. As shown in Figure 4.12a, our MEC implementation is well within the cycle time of a LiDAR sensor, even for the worst-case end-to-end latency in the high density case. On the contrary, the cloud-based implementation of the CA service does not guarantee the 60 ms bound, meaning that the car can potentially act upon obsolete information.

4.4.3 CA service performance

Thanks to the ground truth we built with the SUMO error-log, we checked the performance of our CA service in terms of timely detected collisions. The results of our experimental measurements are reported in Figure 4.13. We can see that both MEC-

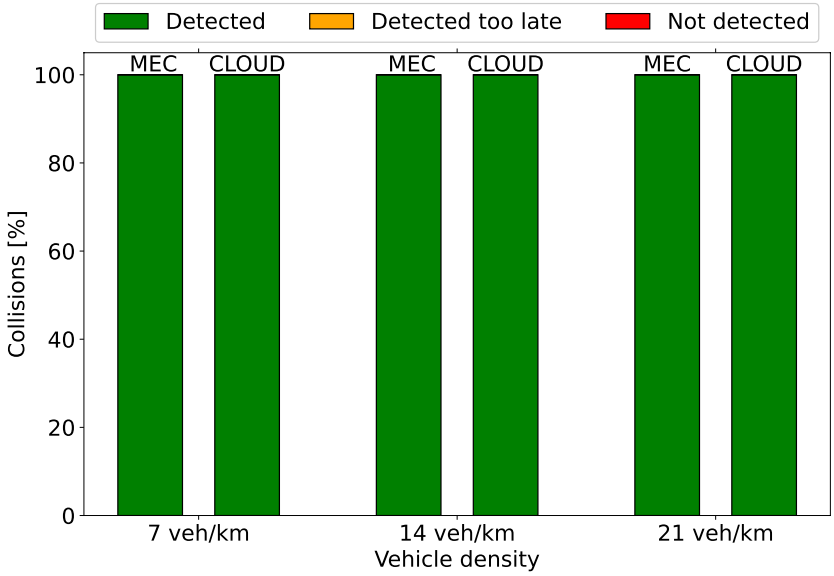
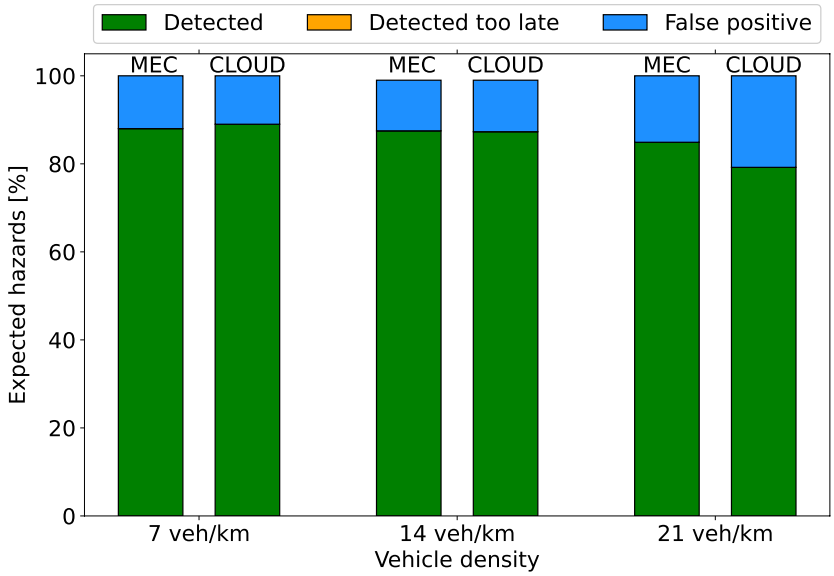


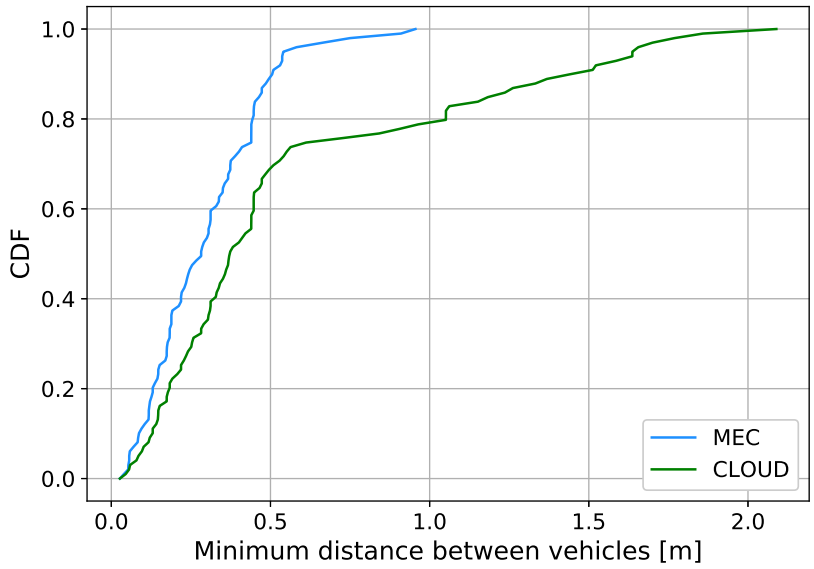
Figure 4.13: Percentage of collisions detected and undected: MEC vs. cloud.

and cloud-based service could alert in time all vehicles on a collision course, and that all crashes were avoided, under all the vehicle densities considered. We hence demonstrated, also through a hardware-in-the-loop simulation technique, the effectiveness of our CA algorithm, which showed excellent reliability.

We then analyzed the number of alarms raised unnecessarily by the CA service. It is worth stressing that false positives can be harmful to the effectiveness of the CA service; indeed, too many false positives annoy the drivers and increase the likelihood they will not react appropriately to future warnings. The analysis of the false positive cases are shown in Figure 4.14. In particular, the percentage of false positives obtained by the MEC and the cloud implementation of the CA service is reported in Figure 4.14a. For the low and medium vehicle density, the number of unnecessary DENMs is similar



(a) Percentage of false positives: MEC vs. cloud.



(b) Distances between vehicles receiving false positive alert messages: MEC vs. cloud.

Figure 4.14: Analysis of the false positives: MEC vs. cloud.

for the two case studies, around 10%. On the contrary, when many vehicles populate the monitored area, with the MEC-based solution we obtained a significantly smaller number of false positives. Figure 4.14b instead, depicts the minimum distance between vehicles involved in situations that led to false positives. Indeed, we were interested

in figuring out whether false positives referred to potentially dangerous situations (i.e., vehicles do not collide but pass very close), or they were due to incorrect detection of the CA algorithm code. As noticeable, each false positive case refers to situations where the vehicles reached a minimum distance lower than 2.1 m. In particular, for the MEC implementation, such a value falls to 1 m. This difference is due to the network latency because the additional delay suffered by the cloud version of our service causes a greater inaccuracy on the calculation of vehicles trajectories.

4.5 Field tests with real vehicles

In order to prove the effectiveness of the CA service in real-word scenarios, several tests with actual cars were performed. These trials were conducted in a test circuit located in the headquarters of CRF-FCA in Beinasco and involved two vehicles and expert drivers, who were instructed to simultaneously approach a junction, creating a collision risk situation.

Firstly, we deployed our CA service in a host, located in a RSU within the test circuit, and, on a separate device, the OAI-based eNB. The equipment of the two vehicles is represented in Figure 4.15. The two cars were equipped with a Uu interface, used for

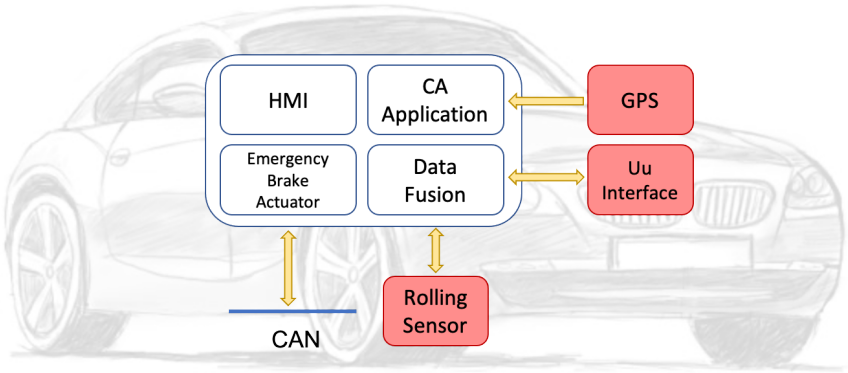


Figure 4.15: Vehicle equipment used in the field tests [6].

communicating with the eNB, and thus to send CAMs and receive DENMs. Position, speed and acceleration are the main inputs to our CA algorithm, and they were provided by a high-accuracy GPS module and coded inside each transmitted CAM.

When a DENM was received, it was decoded and processed by the on-board computer. In our case, the DENM was an alert sent to report an imminent danger, therefore, via the Controller Area Network (CAN) bus, a message was sent toward the *emergency brake actuator*, which triggered an autonomous braking operation to stop the vehicle.

The reception of an alert message was also displayed to the driver thanks to the vehicle HMI.

In each test, vehicles approached the intersection at a speed around 13.89 m/s (i.e., 50 km/h) and they transmitted CAMs at a frequency of 10 Hz. These messages, through the MEC-enabled EPC, were then redirected toward the CIM. Our algorithm was hence always aware of the speed and position of the two vehicles. Once the algorithm evaluated that in a time lower than $t2c_t$ the two vehicles would have reached a distance lower than $s2c_t$, it generated and forwarded a DENM to both vehicles. As described above, the reception of DENMs activated the emergency braking system making the vehicles stop before a potential collision.

The outcome of these test sessions was excellent, as all the collisions were avoided thanks to the timely reception of the DENM messages. Further details together with the collected metrics cannot be reported in this manuscript since they are under non-disclosure agreement.

4.6 Final remarks

In this chapter, we presented the implementation of our C-V2I CA service in a MEC-based architecture. The service showed high reliability and effectiveness in all the scenarios considered, being able to detect each occurred collision. We compared two diverse service implementations: at the MEC platform and in a cloud data-center.

Given the automotive ultra-low latency requirements, we have confirmed that the MEC is undoubtedly one of the main key enablers for delay sensitive applications. We have seen that sensors usually refresh their information every 60 ms, and, with a cloud-based service implementation, it is impossible to guarantee a maximum end-to-end delay lower than this value. This means that the information contained in the DENM is not coherent with data collected by on-board sensors.

Furthermore, recently automotive companies are leaning towards an even more stringent end-to-end latency for the road safety applications, i.e., 20 ms [26]. Such latency is clearly hardly achievable with legacy 4G networks, even with the support of a MEC (see Figure 4.10). Therefore, as already pointed out, only 5G networks can fully enable this kind of critical applications. As a matter of fact, 5G is expected to provide end-to-end latency below 2 ms [8].

Chapter 5

Service Instantiation, Arbitration and Scaling in the 5G-Transformer Architecture

While the previous chapters have described the design of a trajectory-based CA algorithm and the implementation of the corresponding service in a MEC testbed, in this chapter we present the third and last step of our work, i.e., the deployment of the service inside the 5GT platform, together with the evaluation of two of its key network functions.

The 5GT architecture facilitates the instantiation of services by vertical industries and provides runtime management functions to ensure the fulfillment of the requirements throughout the whole lifecycle of the applications. By interacting with the 5GT-VS, a vertical user can request the instantiation of services, which, through the 5GT-SO, can be deployed on the 5GT-MTP. In our case, the 5GT-MTP is represented by the MEC platform, the vertical user is represented by an automotive industry, while the services deployed on the platform are a high priority service, i.e., the CA application, and a low priority one, i.e., a video streaming application.

In the next sections, we describe the service instantiation in the 5GT architecture and we focus on two key network functions available as part of the platform, namely, the service *arbitration* and the service *scaling*.

5.1 Vertical service instantiation

Whenever a vertical industry needs to deploy a service on the 5GT platform, it interacts with the 5GT-VS, which is the system entry point. The 5GT-VS provides a frontend through which verticals can request services and specify their composition in terms of VNFs and performance requirements, by filling in the vertical service descriptor (VSD). Such a VSD is then translated by the 5GT-VS into a network service descriptor (NSD),

which is a service graph composed of VNFs chains and other pivotal instantiation parameters (e.g., deployment flavor). Then, the 5GT-SO interacts with the 5GT-MTP and, by means the NSD, it instantiates the service. The 5GT-MTP is hence responsible for managing the computing, storage and networking resources used to implement the VNFs, and the required transport resources to interconnect the different VNFs of a service placed in diverse points of presence (PoPs), as demanded by the 5GT-SO.

5.1.1 CA service instantiation

In this section we describe the automated deployment of the CA service in the 5GT platform. Figure 5.1 presents the setup used for this demonstration, which was split

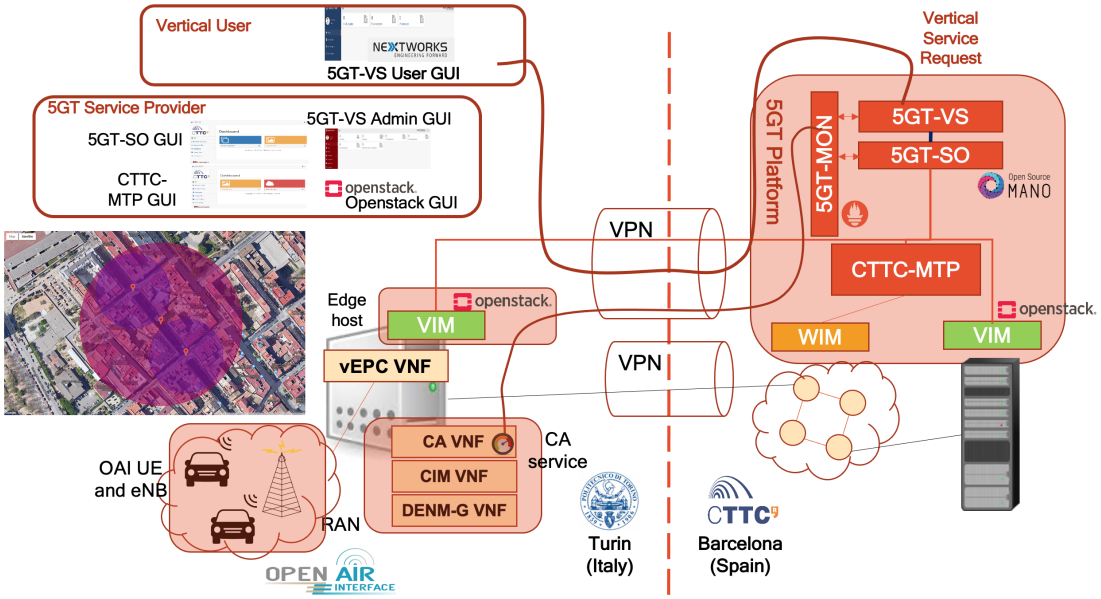


Figure 5.1: Experimental setup used to instantiate the CA service on the 5GT platform.

in two geographical sites, namely, Barcelona (Spain) and Turin (Italy). The 5GT-VS, the 5GT-SO and the 5GT-MON were placed in Barcelona; the 5GT-MTP (i.e., the OAI-based MEC platform), on which the CA service was deployed, in Turin. Differently from the implementation described in the previous chapter, here the DENM-G module was separated from the CA VNF and constituted a third virtual function. As a result, the CA service was composed of the following VNFs:

- **CIM VNF**: it was in charge of receiving, decoding and storing CAMs;
- **CA VNF**: it queried CAMs to the CIM, ran the CA algorithm and, in case of danger, triggered the generation of DENMs to the DENM-G VNF;
- **DENM-G VNF**: it generated and transmitted DENMs messages.

The information coded in the DENMs were sent to the DENM-G by the CA VNF in JSON format. The VIM of the MEC platform was based on an OpenStack all-in-one Devstack deployment, which was visible by the 5GT-SO thanks to a virtual private network (VPN) that connected the two sites.

In order to deploy our CA service using the 5GT architecture, we interacted with the graphical user interface (GUI) of the 5GT-VS (depicted in Figure 5.2) and made the CA service instantiation request. Whenever a vertical user submits an instantiation

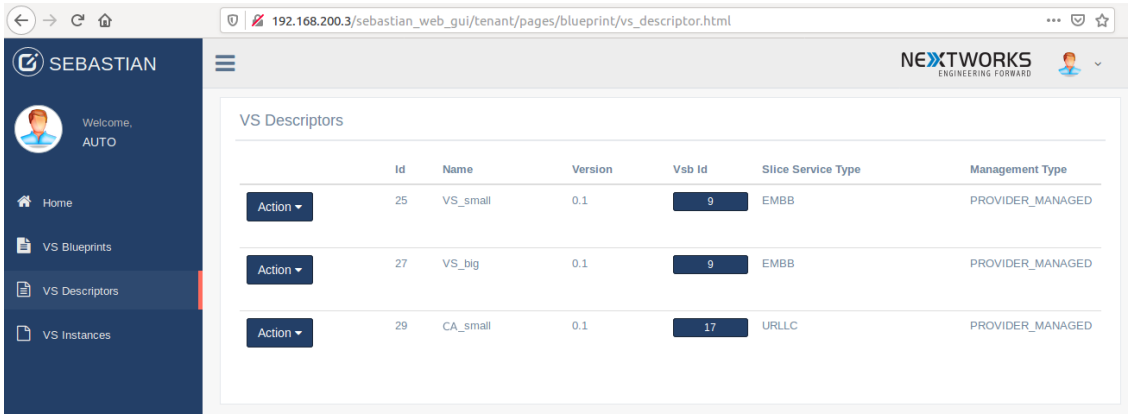


Figure 5.2: 5GT-VS GUI: list of services the automotive vertical can deploy.

request, it is forwarded to the 5GT-SO which calculates the appropriate placement of the different VNFs (in our case, the MEC platform) to honor the requirement embedded in the request and communicates with the MANO platform to launch the different VMs at the underlying VIM (in our case, the Openstack environment). Then, it checks and manages the need for setting up potential links for VNFs placed in different PoPs. In addition to this, since our NSD contained monitoring parameters and autoscaling rules, the 5GT-SO interacted with the 5GT-MON to create the appropriate exporters and alerts to react in front of a violation of the service requirements.

We repeated this instantiation procedure several times and, on average, it took about a couple of minutes to create the three VNFs composing the CA service and the necessary virtual links (VLs). Figure 5.3 shows the virtual network (VN) of the CA service created on our 5GT-MTP (overview provided by the GUI of the 5GT-SO).

5.2 The service *arbitration* function

In this section, we focus on the arbitration capability of the 5GT platform. In this context, arbitration refers to handling the various services, belonging to a same vertical customer, according to their *(i)* SLA requirements, *(ii)* service priorities, and *(iii)* resource budget available to the vertical. In particular, we present a case in which, a low priority video service of the automotive vertical is terminated, when a high priority

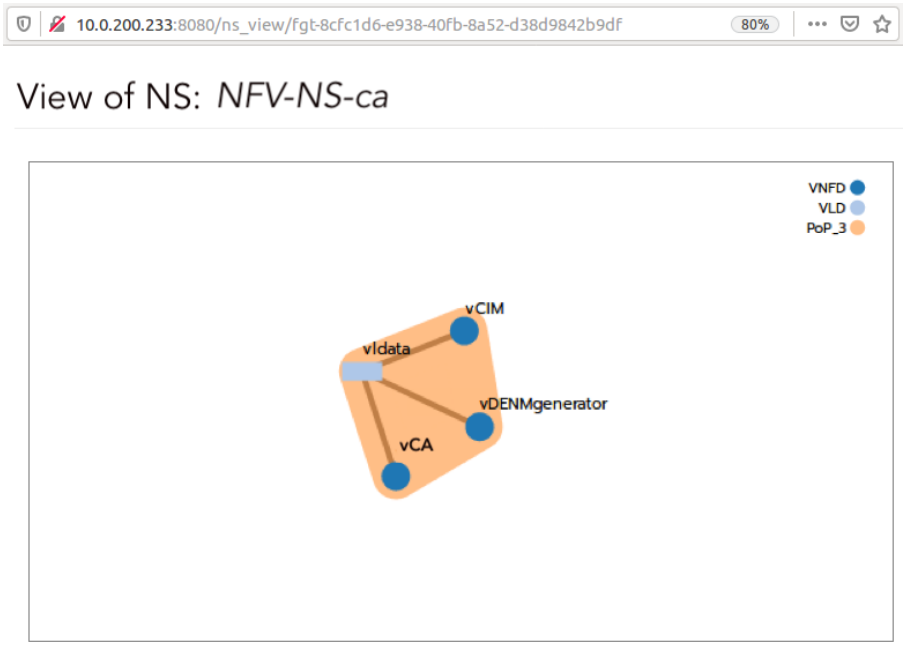


Figure 5.3: 5GT-SO GUI: VN of the CA service.

road safety service needs to be instantiated and there are not enough resources to run both services in parallel.

5.2.1 The 5GT Arbitrator model

One of the main components of the 5GT-VS is the *Arbitrator*. The Arbitrator regulates, for each vertical, how its services get access to the available resource budget, and decides how services are mapped to isolated or shared 5GT slices. The Arbitrator does not have a complete view of the resources available at the infrastructure but works with limited information. It knows the SLAs among the verticals and it may balance the resources assigning probabilities to the services to be deployed based on the SLAs of the verticals [25]. Furthermore, when resources are overused and new services cannot be instantiated, the Arbitrator reassigns the resources among the verticals, according to their SLAs.

The main tasks of the Arbitrator can be summarized as follows:

- Making decisions about how to map new vertical services in network slice istances (NSIs), allowing multiple vertical services to share one or more NSIs or network slice subnet instances (NSSIs);
- Determining the deployment flavor (DF) of each service, ensuring the vertical's quality of service (QoS) requirements, while taking into account the services' priority level.

Finally, note that, upon each new instantiation request, the Arbitrator may need to update the DFs of previously allocated VSIs.

5.2.2 Service arbitration demonstration

In this section, we demonstrate how the 5GT Arbitrator is able to ensure the automated fulfillment of the established business SLAs between a vertical and the provider of the 5GT platform. For this demonstration we used the 5GT platform and the setup shown in Figure 5.4. The latter is a bit more complex than the one previously presented, since

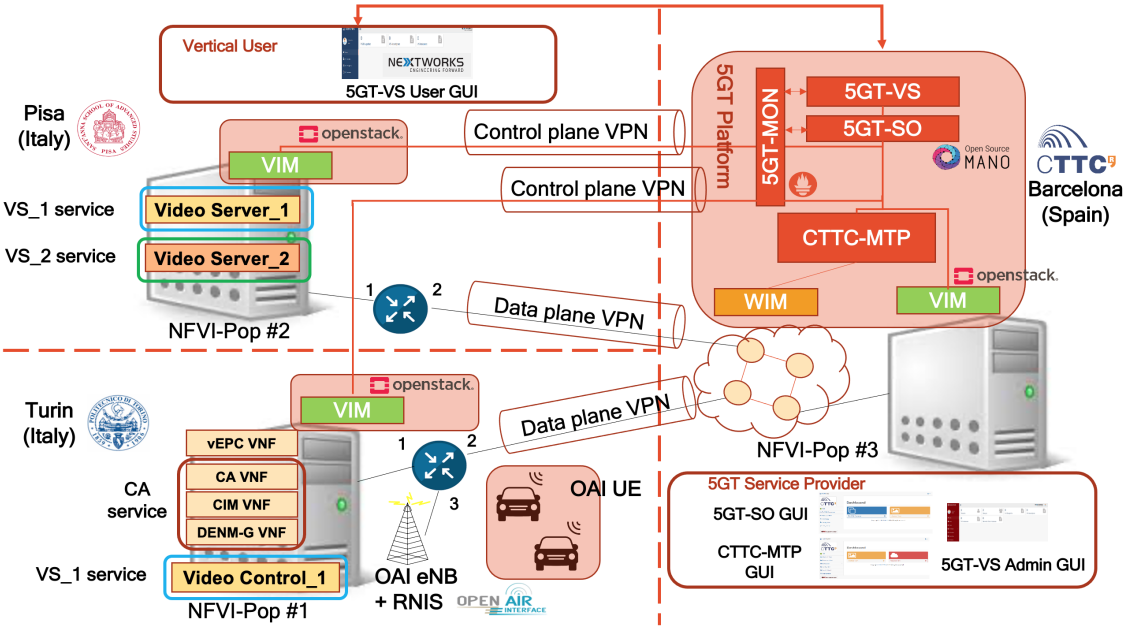


Figure 5.4: Experimental setup used to evaluate the arbitration function of the 5GT platform.

a third NFV infrastructure (NFVI) PoP is added. Again, the 5GT platform (except the 5GT-MTP) is placed in Barcelona, while services are deployed in the NFVI of Turin and Pisa. The three sites are interconnected through VPNs, which are used for both the control and data plane. Each of the three NFVI-PoPs is managed by its own VIM, based on an OpenStack all-in-one Devstack deployment. The Wide Area Infrastructure Manager (WIM) leverages the Ryu open source SDN Controller [87].

For this demonstration we considered two types of vertical services: the well-known CA service and a video streaming service. As described in Section 4.2.3, two UEs acted as applications consumers and the communication between them and the services happened through an OAI eNB. On a dedicated machine, connected to the eNB, we developed a VM running the RNIS. From the eNB, it periodically received the CQI of each active flow and shared this information with the running applications. The OAI-based testbed was deployed in the site of Turin.

The CA service enhances road safety, while the video streaming provides multimedia content to vehicular users. The latter service was designed with two different DFs, having different priorities, specified at the moment of on-boarding through the 5GT-VS:

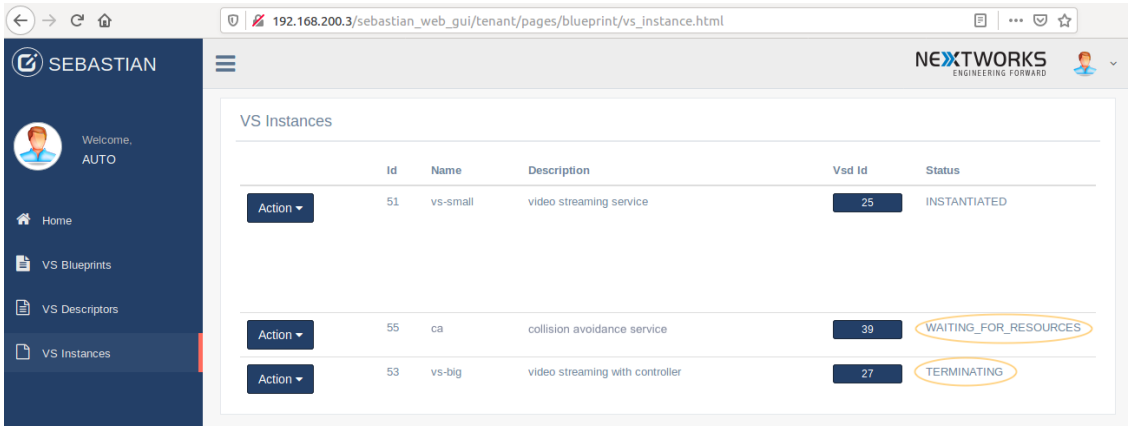
- The *high priority* DF: it consisted of a single VNF (i.e., a video server), holding both the web-server and a cache of videos. It is indicated as *Video_Server_2* in Figure 5.4;
- The *low priority* DF: it was composed of two VNFs, namely, a video server VNF (*Video_Server_1*) and a video controller VNF (*Video_Control_1*). The latter interacted with the RNIS to retrieve the per-UE CQI in order to adjust the quality of each video streaming flow.

The second flavor has lower priority because we considered the primary goal of the automotive vertical to offer the video streaming service, no matter which quality can be provided. The CA service, instead, was composed of the well-known VNFs. Clearly, being a safety service, the CA application has high priority.

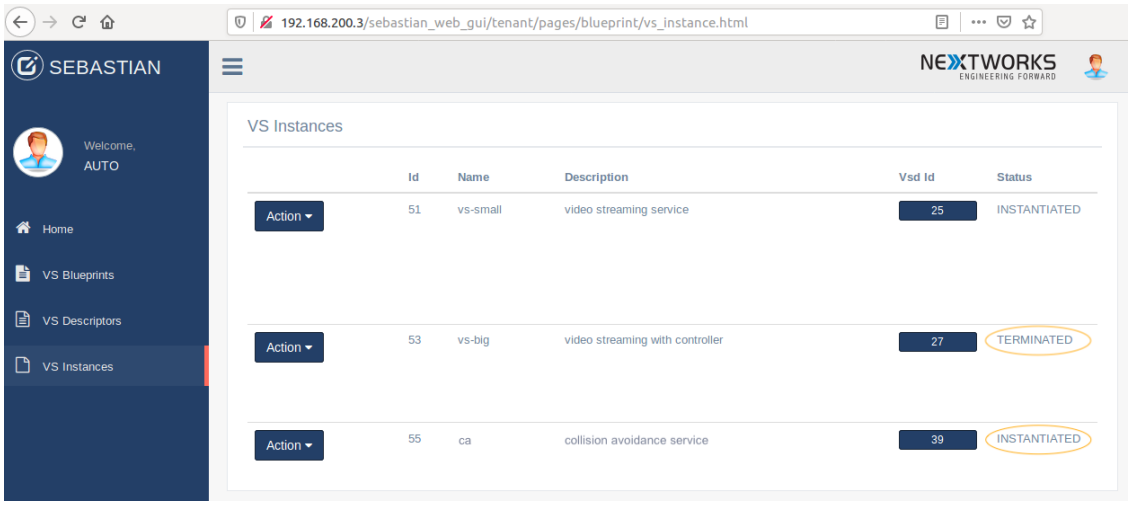
The first service instantiated on the 5GT platform was the entertainment service. Through the 5GT-VS GUI we hence made the request for the instantiation of both video streaming service flavors. According to the procedure described in the previous section, the 5GT-SO processed the request and deployed the video streaming services in the underlying infrastructure while satisfying their constraints embedded in the descriptors. The *low priority* instance was deployed between two PoPs: the two video servers ran in Pisa, whereas the video controller was placed in Turin, close to the RNIS as it needed to leverage low-latency channel quality measurements. Completed the instantiation, the videos hosted in the cache of the two video servers could be consumed by the simulated vehicles (i.e., the two OAI UEs).

We then requested the instantiation of our second service, i.e., the CA service. For each new request, the Arbitrator module of the 5GT-VS checks the resource budget available for the vertical user that made the request. In this particular case, we considered our automotive vertical with a very limited budget of resources, and, as a result, the three services could not be run simultaneously. Therefore, according to the priority of the current deployed services and the new incoming request, the Arbitrator decided to terminate the *low priority* video streaming service, in order to release resources for the CA application. When the Arbitrator takes the decision to terminate a service, the 5GT-VS waits for the confirmation of the termination and then proceeds with the instantiation of the new service.

The status of vertical services can be monitored via the 5GT-VS GUI at any time, as depicted in Figure 5.5. In particular, the latter refers to our arbitration demonstration: Figure 5.5a shows the impossibility for the 5GT-VS to instantiate the CA service owing to the lack of available resources; Figure 5.5b represents the following phase, i.e., the *low priority* service is terminated and the CA service is instantiated.



(a) The CA service cannot be instantiated until resources are released.



(b) The *low priority* video streaming service is terminated and the CA service instantiated.

Figure 5.5: 5GT-VS GUI: status of the services.

5.3 The service *scaling* function

The service scaling is a runtime management function necessary to guarantee the service quality requirements during the whole service lifecycle, under any network operational conditions. In particular, in this section, we show an example of *scaling out* operation.

Due to the stringent latency constraints imposed to road safety services, the processing time of the VNFs has to be as low as possible. For our CA service, as described in Section 4.4.2, the higher the number of vehicles in the monitored area, the higher the processing time of the VNF running the CA algorithm (i.e., the CA VNF). On the contrary, the processing time of the CIM VNF is not very sensitive to the density of vehicular traffic, as well as the DENM-G VNF which, as it was conceived, has a low workload.

Therefore, when the number of vehicles increases significantly, the processing time of the CA VNF may exceed our threshold of 5 ms, risking to violate the maximum end-to-end delay. To address this kind of issue and guarantee to the service provider the SLAs at any time, the platform offers the automated service scaling function. In particular, in this demonstration, a second CA VNF is automatically created in order to reduce the workload on the first instance and, consequently, the overall end-to-end delay.

5.3.1 Criteria for the automated scaling out

As discussed above and shown in Figure 4.11, the processing time of each CA VNF depends also on the number of active users, since a higher number of vehicles corresponds to a higher number of comparisons between cars trajectories, and thus to an increase in the processing latency. The processing time of the CA VNF is strongly correlated to the CPU consumption, since, as can be easily guessed, the higher the CPU consumed by the VM running the CA algorithm, the higher its processing time. By analyzing the relationship between the processing time and the CPU consumed by the VM, it is possible, by monitoring only the latter, to determine whether the scaling operation is needed. With this aim, we ran several tests with different vehicle densities (e.g., 16 veh/km, 18 veh/km, etc.) with a twofold purpose: *(i)* measure the CPU consumption of the VM running the CA VNF; *(ii)* collect the processing times of the CA algorithm. By combining the gathered data, we could obtain the plot reported in Figure 5.6. We

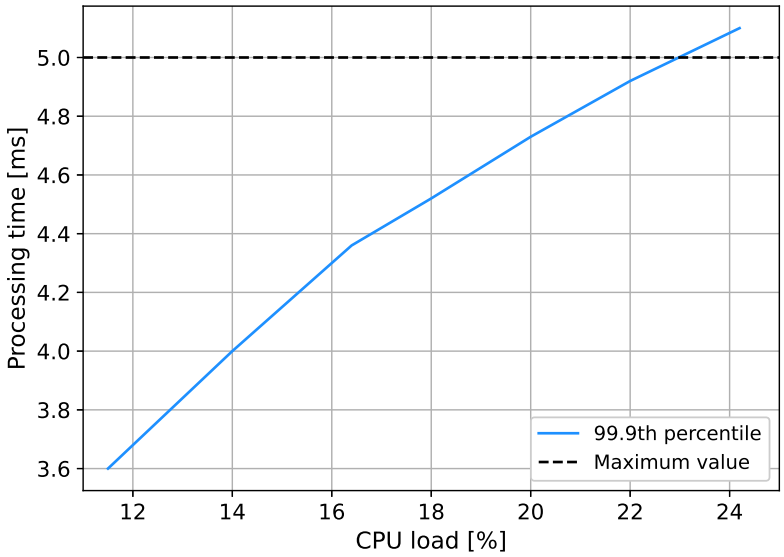


Figure 5.6: Processing time of the CA VNF at different CPU loads.

were interested in the 99.9th percentile, i.e., 99.9% of occurrences for which the CA algorithm is faster than 5 ms when processing the needed car trajectories. As we can see, this could be ensured only if the maximum VM[1] CPU consumption did not exceed 23%.

The monitoring platform of the 5GT architecture (5GT-MON) is based on *Prometheus* [84] to collect information and *Graphana* [52] to display the monitored information in custom and dynamic created dashboards. The 5GT-MON can monitor several metrics, including CPU consumption, RAM usage, and amount of exchanged data. In our scaling out demonstration, the 5GT-MON generated an alert message to the 5GT-SO to create a new CA VNF only when both these conditions were true: *(i)* the overall percentage of CPU usage by the CA VM exceeded the threshold c_t, *(ii)* it remained above the threshold for an amount of time at least equal to t_t seconds. In particular, as soon as the threshold was violated, the 5GT-MON activated an alarm which was turned off only when the CA VNF CPU consumption was again lower than c_t. If this alarm remained active for t_t seconds, then the 5GT-MON sent an alert to the 5GT-SO which managed the scaling out operation.

It was difficult to find the optimal values for the two thresholds c_t and t_t. Indeed, high values do not allow to scale in time (i.e., before the CPU consumption exceeds 23%), whereas low values can trigger a scaling out operation even when not needed. After exhaustive tests, the values that best suited our needs were 15% and 40 s for c_t and t_t, respectively.

5.3.2 Service scaling out demonstration

In this section, we demonstrate the scaling capability offered by the 5GT platform. Our purpose was to increase the workload of the CA VNF so that the latency constraint could not be met, and more resources had to be allocated to the algorithm. In our case, the 5GT platform triggered a *scaling out* operation, i.e., the instantiation of one additional CA VNF. In this way, the coverage region assigned to each instance was smaller, the number of processed CAMs lower, and thus the overall service end-to-end delay reduced.

For this demonstration, we used the setup depicted in Figure 5.1. Once instantiated the CA service through the 5GT-VS, we ran a test similar to the ones described in Section 4.2. We hence used two different OAI UEs to generate the transmission of CAMs coming from vehicles traveling in the monitored region. CAMs were processed by the CA algorithm in order to detect in advance possible collisions and generate the corresponding DENMs. In parallel, the CPU consumption of the CA VNF was continuously monitored by the 5GT-MON platform to guarantee the low latency requirement of the service, that is, as the reader knows, verifying that the processing time of the CA VNF is below the 5ms-threshold.

[1]The CA VNF was virtualized in a VM with 1 core at 2 GHz and 4 GB RAM.

The input traces used by the two VehicleSimulator instances were prepared so that the number of simulated vehicles grew over time. In this way, as the test progressed, the number of CAM messages parsed by the CA algorithm increased, so its CPU consumption. When the latter exceeded the threshold c_t for t_t seconds, the 5GT-MON sent an alert to the 5GT-SO to trigger the scaling out operation. The 5GT-SO hence deployed a new instance of the CA VNF on our NFVI-PoP in Turin.

After the autoscaling operation, the CA service included two CA VNFs and the monitored scenario was split in two areas: one under the control of the first instance, the other under the control of the second. The center of the two areas coincided with the center of the two crossroads (see Figure 4.9). In this way, the CPU load of both instances guaranteed a processing time lower than 5 ms, satisfying our service requirements. Figure 5.7 shows the CPU consumption of the two CA VNFs: the chart at the top refers to the first instance, the one at the bottom right to the second. The second CA VNF starts at 19:18:30[2], and, at the same time instant, it is possible to see that the CPU consumption of the first CA VNF starts decreasing, dropping from 20% to 12%.

Figure 5.7: Grafana GUI: percentage of CPU consumption of the two CA VNF instances.

[2]100% of CPU consumption before this time instant is due to operations performed by the CA VM at boot.

5.4 Final remarks

In this section, we described the instantiation of our CA service on the 5GT architecture, and the assessment of two of the most important network functions for 5G and next generation mobile networks, namely, arbitration and scaling.

In our tests, the platform has confirmed to be very effective in enabling vertical users to easily deploy their tailored services. The service instantiation has proven to be very simple thanks to the easy-to-use interface provided by the 5GT-VS, as well as fast (in the case of our CA service, about 2 minutes were required).

The runtime management functions offered by the platform are pivotal for supplying safety-critical services, such as road safety applications, which have stringent requirements. In our tests, the arbitration capability of the 5GT-VS always guaranteed the provision of the services with high priority, terminating the ones at lower priority when the resources were not sufficient to run all of them in parallel. We also tested the automated scaling capability, focusing on the scaling out operation. It allowed to automatically deploy a new instance of the CA VNF when the workload (i.e., the number of CAMs to be processed) could not be efficiently handled anymore by a single VNF, without causing an unacceptable increase in the end-to-end latency. The instantiation of the second CA VNF was triggered by the increase in the CPU consumption of the first instance, which was continuously monitored by the 5GT-MON component.

Finally, it is worth emphasizing an aspect. On the 5GT platform, the CIM, the CA and the DENM-G VNFs were virtualized as VMs within the Openstack environment. They could be virtualized through Docker containers since, as shown in our study, Docker guarantees excellent performance and a low virtualization overhead. However, since the only VIM compatible with the 5GT architecture was Openstack, which does not handle Docker containers, we were forced to select the hypervisor-based technology to virtualize our CA service.

Chapter 6

Conclusions

Network Function Virtualization (NFV), network slicing and Multi-access Edge Computing (MEC) are three pillar technologies for 5G and for the next generation of mobile networks. They enable ultra-high data rates, ultra-low latency, and guarantee support to a wide range of vertical services with strict and heterogeneous requirements.

In this thesis we have presented a particular automotive service, i.e., a (cellular-V2I-based) collision avoidance (CA) service. It leverages the exchange of periodic and anonymous Cooperative Awareness Messages (CAMs) between road users and a CA algorithm hosted in the network infrastructure. The latter, by combining the information contained in these messages, is able to detect in advance any imminent danger that may result in a collision, and warn the involved drivers. In order to guarantee the low latency required by safety services, we were interested in a MEC-based implementation of the CA application.

The core of our automotive service is the CA algorithm presented in Section 3.2. It is a trajectory-based algorithm, suitable for the detection of any type of possible collisions, including the ones involving vehicles and vulnerable road users, such as pedestrians and bikers. It takes as input position, speed, acceleration and heading of two road users and determines whether they are on a collision course. After having designed the algorithm, due to the lack of suitable simulation tools, we built our simulator (based on the well-known and open-source SimuLTE-Veins) in order to asses its effectiveness. The algorithm provided excellent results, being able to timely detect both vehicle-with-vehicle and vehicle-with-pedestrian collisions.

By leveraging the CA algorithm, we then built a first real-word testbed of a CA application on an OAI architecture including MEC functionalities. The CA service was composed of two main VNFs running on top of the MEC host: the so-called Cooperative Information Manager (CIM) VNF, decoding and storing the CAMs sent by the vehicles, and the CA VNF, gathering the CAMs from the CIM, running the CA algorithm and transmitting the Decentralized Environmental Notification Messages (DENMs) in case

of possible collisions. In Section 4.4 we have presented our service performance assessment through a hardware-in-the-loop simulation technique. We compared our MEC-based solution against a cloud-based implementation of the CA service, i.e., with the two VNFs running in a cloud data-center. Both service implementations have proven to be reliable, timely detecting all the collisions.

However, a couple of points are worth mentioning. First, in highly congested traffic scenarios, we have noticed a higher number of false positives when the service was running in the cloud. False positives can negatively affect the effectiveness of the CA service as much as false negatives; indeed, too many false positives may annoy the drivers and increase the likelihood they will not react appropriately to future warnings or disable the system altogether. Second, LiDAR sensors typically refresh their information every 60 ms. This means that the information contained in the DENMs is coherent with on-board sensors data only if the end-to-end latency does not exceed such a value. Our measurements have demonstrated that the MEC implementation can guarantee a latency which is well within the cycle time of LiDAR sensors, even for the worst-case end-to-end latency, in the high density traffic scenarios. On the contrary, the cloud-based approach has constantly violated the 60 ms bound. In other words, when the CA service is running in the cloud, a vehicle cannot correctly process together the information collected by the ADAS sensors and the one received with the DENMs, as the latter becomes obsolete.

Given that, in the scenarios considered for the evaluation, the totality of the collisions was timely detected and considering the analysis described above, we have proven how reliability and latency requirements can be successfully met when the automotive domain is assisted by the cellular network. This is especially true when such a network includes the availability of computational capabilities at the edge, which enables a substantial reduction in network latency. Moreover, as detailed in Section 3.4, our centralized approach allows to extend the service also to vulnerable road users, which, through their smartphones, can send CAMs and receive DENMs. This is indeed not possible with traditional distributed systems (owing to the reduced capabilities of smartphones when compared to vehicular on-board units), which are not able to quickly process all the CAMs they may receive and, timely warn the road users in case of danger.

Due to the growing interest in *safety* applications, our CA service was selected as one of the use cases within the 5G-Transformer (5GT) European project. The 5GT project has proposed and developed an open and flexible 5G transport and computing platform tailored to support the heterogeneous service requirements of several industry segments. With our CA service, we tested both the capability of the platform in terms of deploying services when required by vertical industries, as well as two fundamental management functions provided at service runtime, i.e., arbitration and scaling. The platform, through its easy-to-use interface, allows vertical users to quickly instantiate their services. Furthermore, both service arbitration and scaling are able to guarantee the fulfillment of the requirements of the CA service, throughout its whole lifecycle. In conclusion, the 5GT platform perfectly addresses the need for a flexible network

architecture, on which vertical industries can deploy services with strict requirements, which will be, in turn, always guaranteed.

In our experimental tests, due to hardware and software availability, we leveraged the LTE technology to enable the communication between the simulated vehicles and the CA service. A potential extension of this work could be the implementation of this service in a full 5G standalone testbed, or even better, in a heterogeneous scenario where both the 4G and 5G networks are supported. In such a way it could be possible to directly compare the service performances when the two different technologies are used, or make analysis considering enabled at the same time both the access technologies. It is worth emphasizing that both the CA application and the 5GT architecture are independent of the underlying communication technology: they are ready to be ported to a full 5G standalone implementation, without any modification. Clearly, we expect from a 5G network, higher and better performance than the one assessed in this work, thanks to a sharp reduction in the latency between the CAM transmission and DENM reception, and a very high reliability (at least 99.999%), which is currently hardly achievable with 4G. This consideration is also tightly related to the fact that automotive companies are leaning towards a very stringent end-to-end latency, within 20 ms, which cannot be ensured by the LTE technology, neither with the support of the MEC.

Other future research directions could involve the design, implementation and performance assessment of other road safety applications, such as Emergency Vehicle Warning and Cooperative Lane-Merging. We believe that, among the wide range of vehicular applications, safety services are undoubtedly the most important and the ones on which, academia and research centers, should invest a great deal of attention. Vehicular communication has the potential to enable many safety applications, which would have a great impact on our lives, as they could dramatically reduce the number of fatalities and serious injuries in road accidents (halving this number, according to a recent study of the European Commission [14]). Therefore, the development and analysis of mobile safety applications and the design of 5G ready systems play a crucial role for the scientific community, which has the opportunity to revolutionize the way we travel, making it much safer.

Appendix A

Published and Submitted Content

Below, the list of my published papers during my Ph. D. program:

[17] **Giuseppe Avino**, Marco Malinverno, Francesco Malandrino, Claudio Casetti, Carla Fabiana Chiasserini. "Characterizing Docker Overhead in Mobile Edge Computing Scenarios". Published in *Proceedings of the Workshop on Hot Topics in Container Networking and Networked Systems, p. 30-35*, 21-25 August 2017, Los Angeles, CA, USA. https://doi.org/10.1145/3094405.3094411

[18] **Giuseppe Avino**, Marco Malinverno, Francesco Malandrino, Claudio Casetti, Carla Fabiana Chiasserini, Giovanni Nardini, Salvatore Scarpina. "Poster: A Simulation-based Testbed for Vehicular Collision Detection". Published in *the IEEE Vehicular Networking Conference (VNC)*, 27-29 November 2017, Turin, Italy. https://ieeexplore.ieee.org/document/8275655

[65] Marco Malinverno, **Giuseppe Avino**, Claudio Casetti, Carla Fabiana Chiasserini, Francesco Malandrino, Salvatore Scarpina. "Performance Analysis of C-V2I-based Automotive Collision Avoidance". Published in *the 19th IEEE International Symposium on "A World of Wireless, Mobile and Multimedia Networks" (WoWMoM 2018)*, 12-15 June 2018, Chania, Greece. https://ieeexplore.ieee.org/abstract/document/8449772

[15] **Giuseppe Avino**, Marco Malinverno, Claudio Casetti, Carla Fabiana Chiasserini, Francesco Malandrino, Marco Rapelli, Giuliana Zennaro. "Support of Safety Services through Vehicular Communications: The Intersection Collision Avoidance Use Case". Published in *the IEEE International Conference of Electrical and Electronic Technologies for Automotive, p. 1-6*, 9-11 July 2018, Milan, Italy. https://ieeexplore.ieee.org/abstract/document/8493191

[62] Francesco Malandrino, Carla Fabiana Chiasserini, **Giuseppe Avino**, Marco Malinverno, Scott Kirkpatrick. "From Megabits to CPU Ticks: Enriching a Demand Trace in the Age of MEC". Published in *IEEE Transactions on Big Data (2018).* https://ieeexplore.ieee.org/document/8447497

[56] Giada Landi, Pietro Giardina, Marco Capitani, Koteswararao Kondepu, Luca Valcarenghi, **Giuseppe Avino**. "Demo: provisioning and automated scaling of network slices for virtual Content Delivery Networks in 5G infrastructures". Published in *Proceedings of the Twentieth ACM International Symposium on Mobile Ad Hoc Networking and Computing, p. 397-398*, 2-5 July 2019, Catania, Italy. https://dl.acm.org/doi/abs/10.1145/3323679.3326613

[16] **Giuseppe Avino**, Paolo Bande, Pantelis A. Frangoudis, Christian Vitale, Claudio Casetti, Carla Fabiana Chiasserini, Kalkidan Gebru, Adlen Ksentini, Giuliana Zennaro. "A MEC-based Extended Virtual Sensing for Automotive Services". Published in *IEEE Transactions on Network and Service Management, pp.1450-1463*, July 2019. https://ieeexplore.ieee.org/abstract/document/8781832

[20] Jorge Baranda, **Giuseppe Avino**, Josep Mangues-Bafalluy, Luca Vettori, Ricardo Martínez, Carla Fabiana Chiasserini, Claudio Casetti, Paolo Bande, Marina Giordanino, Marco Zanzola. "Automated deployment and scaling of automotive safety services in 5G-Transformer". Published in *the IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN), p. 1-2*, 12-14 November 2019, Dallas, TX, USA. https://ieeexplore.ieee.org/abstract/document/9039990

[64] Marco Malinverno, **Giuseppe Avino**, Claudio Casetti, Carla Fabiana Chiasserini, Francesco Malandrino, Salvatore Scarpina. "MEC-based Collision Avoidance for Vehicles and Vulnerable Users". Published in *IEEE Vehicular Technology Magazine, vol. 15, no. 1, pp. 27-35*, March 2020. https://doi.org/10.1109/MVT.2019.2953770

[19] Jorge Baranda, Josep Mangues-Bafalluy, Luca Vettori, Ricardo Martínez, **Giuseppe Avino**, Carla Fabiana Chiasserini, Corrado Puligheddu, Claudio Casetti, Juan Brenes, Giada Landi, Koteswararao Kondepu, Francesco Paolucci, Silvia Fichera, Luca Valcarenghi. "Demo Abstract: Arbitrating Network Services in 5G Networks for Automotive Vertical Industry". Published in *the IEEE Conference on Computer Communications Workshops (INFOCOM WORKSHOPS), p 1318-1319*, 6-9 July 2020, Toronto, ON, Canada. https://ieeexplore.ieee.org/document/9162679

List of acronyms

3GPP	Third Generation Partnership Project
5G	Fifth Generation Mobile Networks
5GT	5G-Transformer
ADAS	Advanced Driver-Assistance Systems
API	Application Programming Interface
AppD	Application Descriptor
BS	Base Station
BSS	Business Support System
CA	Collision Avoidance
CAM	Cooperative Awareness Message
CDF	Cumulative Distribution Function
CFS	Customer Facing Service
CPU	Central Processing Unit
CQI	Channel Quality Indicator
CUPS	Control and User Plane Separation
C-V2I	Cellular-V2I
C-V2X	Cellular-V2X
DENM	Decentralised Environmental Notification Message
DENM-G	DENM-Generator
DF	Deployment Flavor
EHF	Extremely High Frequency
eNB	Evolved Node B
EPC	Evolved Packet-Core
ETSI	European Telecommunications Standards Institute
GPRS	General Packet Radio Service

GPS	Global Positioning System
GTP	GPRS Tunneling Protocol
HMI	Human-Machine Interface
HSS	Home Subscriber Server
HTTP	HyperText Transfer Protocol
IM	Information Manager
ISG	Industry Specification Group
IT	Information Technology
ITS	Intelligent Transport Systems
ITU	International Telecommunication Union
KPI	Key Performance Indicator
LDM	Local Dynamic Map
LiDAR	Light Detection And Ranging
LoS	Line-of-Sight
LTE	Long Term Evolution
LTE-A	LTE-Advanced
LXC	LinuX Containers
MANO	MANagement and Orchestration
ME	Mobile Edge
MEC	Multi-access Edge Computing
MEMS	Micro-Electro-Mechanical Systems
MEP	MEC Platform
MEPM	MEP Manager
MME	Mobility Management Entity
mmWave	Millimiter Wave
MNO	Mobile Network Operator
MON	MONitoring platform
MTP	Mobile Transport Platform
MVNO	Mobile Virtual Network Operator
NF	Network Function
NFV	Network Function Virtualization

NFVI	Network Function Virtualization Infrastructure
NSD	Network Service Descriptor
NSI	Network Slice Istance
NSSI	Network Slice Subnet Instance
OS	Operating System
OSS	Operating Support System
PID	Process IDentifier
PoP	Point of Presense
QoS	Quality of Service
RAN	Radio Access Network
REST	REpresentational State Transfer
RNIS	Radio Network Information Service
SLA	Service Level Agreement
SO	Service Orchestrator
SP	Service Provider
STDERR	STandarD Error
STDIO	STandarD Input-Output
S/P-GW	Service/Packet-Gateway
S/P-GW-C	S/P-GW- Control Plane
S/P-GW-U	S/P-GW- User Plane
UC	Use Case
UE	User Equipment
UPER	Unaligned Packet Encoding Rule
VA	Virtual Application
VF	Virtual Function
VIM	Virtual Infrastructure Manager
VL	Virtual Link
VN	Virtual Network
VNF	Virtual Network Function
VNFD	VNF Descriptor
VNFFG	VNF Forwarding Graph

VS	Vertical Slicer
VSD	Vertical Service Descriptor
VPN	Virtual Private Network
VRU	Vulnerable Road User
V2I	Vehicle-to-Infrastructure
V2N	Vehicle-to-Network
V2P	Vehicle-to-Pedestrian
V2V	Vehicle-to-Vehicle
V2X	Vehicle-to-Everything
WIM	Wide area Infrastructure Manager

Bibliography

[1] *3GPP TR 21.914 V14.0.0 - 3rd Generation Partnership Project; Technical Specification Group Services and System Aspects; Release 14 Description; Summary of Rel-14 Work Items (Release 14)*. Technical Requirement. 3rd Generation Partnership Project, 2018.

[2] *3GPP TR 21.915 V15.0.0 - 3rd Generation Partnership Project; Technical Specification Group Services and System Aspects; Release 15 Description; Summary of Rel-15 Work Items (Release 15)*. Technical Requirement. 3rd Generation Partnership Project, 2019.

[3] *3GPP TR 21.916 V0.6.0 - 3rd Generation Partnership Project; Technical Specification Group Services and System Aspects; Release 16 Description; Summary of Rel-16 Work Items (Release 16)*. Technical Requirement. 3rd Generation Partnership Project, 2020.

[4] *5G-Transformer*. Visited on 2020-10-30. URL: http://5g-transformer.eu/.

[5] 5G-TRANSFORMER. *D1.2, 5G-TRANSFORMER initial system design*. Tech. rep. May 2018.

[6] 5G-TRANSFORMER. *Report on vertical requirements and use cases*. Tech. rep. Dec. 2017.

[7] *5T Open data repository*. Visited on 2020-11-20. URL: http://www.5t.torino.it/open-data/.

[8] NGMN Alliance. "5G white paper." In: *Next generation mobile networks, white paper* (2015), pp. 1–125.

[9] NGMN Alliance. "Description of network slicing concept." In: *NGMN 5G P* 1.1 (2016).

[10] NGMN Alliance. "Perspectives on vertical industries and implications for 5G." In: *White Paper, Jun* (2016).

[11] NGMN Alliance. "V2X white paper." In: *White Paper* 1 (2018).

[12] *Amazon Cloud*. https://calculator.s3.amazonaws.com/index.html. Visited on 2020-11-24.

[13] Giuseppe Araniti et al. "LTE for vehicular networking: a survey." In: *IEEE communications magazine* 51.5 (2013), pp. 148–157.

[14] GA Association et al. "An assessment of lte-v2x (pc5) and 802.11 p direct communications technologies for improved road safety in the eu." In: *5G Automotive Association, Tech. Rep.* (2017).

[15] G Avino et al. "Support of safety services through vehicular communications: The intersection collision avoidance use case." In: *2018 International Conference of Electrical and Electronic Technologies for Automotive*. IEEE. 2018, pp. 1–6.

[16] Giuseppe Avino et al. "A MEC-based extended virtual sensing for automotive services." In: *IEEE Transactions on Network and Service Management* 16.4 (2019), pp. 1450–1463.

[17] Giuseppe Avino et al. "Characterizing docker overhead in mobile edge computing scenarios." In: *Proceedings of the Workshop on Hot Topics in Container Networking and Networked Systems*. 2017, pp. 30–35.

[18] Giuseppe Avino et al. "Poster: A simulation-based testbed for vehicular collision detection." In: *2017 IEEE Vehicular Networking Conference (VNC)*. IEEE. 2017, pp. 39–40.

[19] J. Baranda et al. "Arbitrating Network Services in 5G Networks for Automotive Vertical Industry." In: *IEEE INFOCOM 2020 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*. 2020, pp. 1318–1319. DOI: 10.1109/INFOCOMWKSHPS50562.2020.9162679.

[20] Jorge Baranda et al. "Automated deployment and scaling of automotive safety services in 5G-Transformer." In: *2019 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*. IEEE. 2019, pp. 1–2.

[21] A. Bazzi et al. "On the Performance of IEEE 802.11p and LTE-V2V for the Cooperative Awareness of Connected Vehicles." In: *IEEE Transactions on Vehicular Technology* 66.11 (2017), pp. 10419–10432. DOI: 10.1109/TVT.2017.2750803.

[22] Alessandro Bazzi et al. "On the performance of IEEE 802.11 p and LTE-V2V for the cooperative awareness of connected vehicles." In: *IEEE Transactions on Vehicular Technology* 66.11 (2017), pp. 10419–10432.

[23] Alessandro Bazzi et al. "Survey and perspectives of vehicular Wi-Fi versus sidelink cellular-V2X in the 5G era." In: *Future Internet* 11.6 (2019), p. 122.

[24] Claudia Campolo, Antonella Molinaro, and Riccardo Scopigno. "Vehicular ad hoc Networks." In: *Standards, Solutions, and Research* (2015).

[25] C. Casetti et al. "Arbitration Among Vertical Services." In: *2018 IEEE 29th Annual International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC)*. 2018, pp. 153–157. DOI: 10.1109/PIMRC.2018.8580852.

[26] Claudio Casetti et al. "Arbitration among vertical services." In: *2018 IEEE 29th Annual International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC)*. IEEE. 2018, pp. 153–157.

[27] Giammarco Cecchini et al. "Performance comparison between IEEE 802.11 p and LTE-V2V in-coverage and out-of-coverage for cooperative awareness." In: *2017 IEEE Vehicular Networking Conference (VNC)*. IEEE. 2017, pp. 109–114.

[28] Junil Choi et al. "Millimeter-wave vehicular communication to support massive automotive sensing." In: *IEEE Communications Magazine* 54.12 (2016), pp. 160–167.

[29] *Cloudify Orchestrator Platform*. Visited on 2020-10-30. URL: https://cloudify.co/.

[30] *Commercial LIDAR sensor for vehicle*. https://www.bosch-mobility-solutions.com/en/products-and-services/passenger-cars-and-light-commercial-vehicles/driver-assistance-systems/predictive-emergency-braking-system/mid-range-radar-sensor-(mrr)/. [Online; accessed 18-April-2019]. 2019.

[31] Docker.com. *The Docker Containerization Platform*. Visited on 2020-10-28. URL: https:https://www.docker.com/.

[32] Christian Ehrhardt. "CPU time accounting." In: *Last accessed: Aug* (2013).

[33] EN ETSI. *302 637-2 (V1.3.1).(2014) Intelligent Transport Systems (ITS); Vehicular Communications; Basic Set of Applications; part 2: Specification of cooperative awareness basic service*. Tech. rep. Technical report. Draft ETSI TS.

[34] NFVISG ETSI. *GS NFV-MAN 001 V1. 1.1 Network Function Virtualisation (NFV); Management and Orchestration*. 2014.

[35] TCITS ETSI. "Intelligent transport systems (ITS); vehicular communications; basic set of applications; definitions." In: *Tech. Rep. ETSI TR 102 6382009* (2009).

[36] *Mobile Edge Computing (MEC); Mobile Edge Management; Part 2: Application lifecycle, rules and requirements management*. Group Specification. July 2017.

[37] *Mobile Edge Computing (MEC); Radio Network Information API*. Group Specification. Version 1.1.1. July 2017.

[38] *Ettus USRP B210 board*. Visited on 2020-11-23. URL: https://www.ettus.com/all-products/ub210-kit/.

[39] AE Fernandez, M Fallgren, and N Brahmi. "5GCAR scenarios, use cases, requirements and KPIs." In: *Fifth Generation Communication Automotive Research and innovation, Tech. Rep. D* 2 (2017).

[40] Gerhard P Fettweis. "The tactile internet: Applications and challenges." In: *IEEE Vehicular Technology Magazine* 9.1 (2014), pp. 64–70.

[41] *FFmpeg*. Visited on 2020-11-12. URL: https://ffmpeg.org/.

[42] *FFserver*. Visited on 2020-11-12. URL: https://trac.ffmpeg.org/wiki/ffserver.

[43] Xenofon Foukas et al. "FlexRAN: A Flexible and Programmable Platform for Software-Defined Radio Access Networks." In: *Proc. ACM CoNEXT*. 2016.

[44] *FreeBSD*. Visited on 2020-11-10. URL: https://www.freebsd.org/.

[45] *FRep support*. Visited on 2020-11-12. URL: https://strai.x0.com/frep/.

[46] Laurent Gallo and Jerome Haerri. "Unsupervised long-term evolution device-to-device: A case study for safety-critical v2x communications." In: *IEEE Vehicular Technology Magazine* 12.2 (2017), pp. 69–77.

[47] *Genymotion*. Visited on 2020-11-12. URL: https://www.genymotion.com/.

[48] Marco Giordani, Andrea Zanella, and Michele Zorzi. "LTE and millimeter waves for V2I communications: An end-to-end performance comparison." In: *2019 IEEE 89th Vehicular Technology Conference (VTC2019-Spring)*. IEEE. 2019, pp. 1–7.

[49] Marco Giordani et al. "On the Feasibility of Integrating mmWave and IEEE 802.11 p for V2V Communications." In: *2018 IEEE 88th Vehicular Technology Conference (VTC-Fall)*. IEEE. 2018, pp. 1–7.

[50] Marco Giordani et al. "Performance study of LTE and mmWave in vehicle-to-network communications." In: *2018 17th Annual Mediterranean Ad Hoc Networking Workshop (Med-Hoc-Net)*. IEEE. 2018, pp. 1–7.

[51] Fabio Giust, Xavier Costa-Perez, and Alex Reznik. "Latency Critical IoT Applications in 5G: Perspective on the Design of Radio Interface and Network Architecture." In: *IEEE 5G Tech Focus* 1.4 (Dec. 2017).

[52] *Grafana*. https://grafana.com/. Visited on 2020-11-26.

[53] Michael R Hafner et al. "Cooperative collision avoidance at intersections: Algorithms and experiments." In: *IEEE Transactions on Intelligent Transportation Systems* 14.3 (2013), pp. 1162–1175.

[54] Yun Chao Hu et al. "Mobile edge computing—A key technology towards 5G." In: *ETSI white paper* 11.11 (2015), pp. 1–16.

[55] A. Kivity et al. "kvm: the Linux virtual machine monitor." In: *Proc. Linux Symposium*. 2007.

[56] Giada Landi et al. "Provisioning and automated scaling of network slices for virtual content delivery networks in 5G infrastructures." In: *Proceedings of the Twentieth ACM International Symposium on Mobile Ad Hoc Networking and Computing*. 2019, pp. 397–398.

[57] Xiaoying Lei and Seung Hyong Rhee. "Performance analysis and enhancement of IEEE 802.11 p beaconing." In: *EURASIP Journal on Wireless Communications and Networking* 2019.1 (2019), pp. 1–10.

[58] Y. Li et al. "A Markov Jump Process Model for Urban Vehicular Mobility: Modeling and Applications." In: *IEEE Transactions on Mobile Computing* 13.9 (2014), pp. 1911–1926. DOI: 10.1109/TMC.2013.159.

[59] *Linux Container 1xd Webpage*. https://linuxcontainers.org/lxd/. [Online; accessed 2020-11-22]. 2019.

[60] Pablo Alvarez Lopez et al. "Microscopic Traffic Simulation using SUMO." In: *The 21st IEEE International Conference on Intelligent Transportation Systems*. IEEE, 2018. URL: https://elib.dlr.de/124092/.

[61] Zachary MacHardy et al. "V2X access technologies: Regulation, research, and remaining challenges." In: *IEEE Communications Surveys & Tutorials* 20.3 (2018), pp. 1858–1877.

[62] Francesco Malandrino et al. "From megabits to cpu ticks: Enriching a demand trace in the age of mec." In: *IEEE Transactions on Big Data* (2018).

[63] Francesco Malandrino et al. "Verification and inference of positions in vehicular networks through anonymous beaconing." In: *IEEE transactions on mobile computing* 13.10 (2014), pp. 2415–2428.

[64] Marco Malinverno et al. "Edge-based collision avoidance for vehicles and vulnerable users: an architecture based on MEC." In: *IEEE vehicular technology magazine* 15.1 (2019), pp. 27–35.

[65] Marco Malinverno et al. "Performance analysis of C-V2I-based automotive collision avoidance." In: *2018 IEEE 19th International Symposium on" A World of Wireless, Mobile and Multimedia Networks"(WoWMoM)*. IEEE. 2018, pp. 1–9.

[66] Josep Mangues-Bafalluy et al. "5G-TRANSFORMER Service Orchestrator: design, implementation, and evaluation." In: *2019 European Conference on Networks and Communications (EuCNC)*. IEEE. 2019, pp. 31–36.

[67] *Mobile Edge Computing (MEC); Framework and Reference Architecture*. Group Specification. Jan. 2019.

[68] Rashid Mijumbi et al. "Network function virtualization: State-of-the-art and research challenges." In: *IEEE Communications surveys & tutorials* 18.1 (2015), pp. 236–262.

[69] R. Miller and Qingfeng Huang. "An adaptive peer-to-peer collision warning system." In: *Vehicular Technology Conference. IEEE 55th Vehicular Technology Conference. VTC Spring 2002 (Cat. No.02CH37367)*. Vol. 1. 2002, 317–321 vol.1. DOI: 10.1109/VTC.2002.1002718.

[70] Zeeshan Hameed Mir and Fethi Filali. "LTE and IEEE 802.11 p for vehicular networking: a performance evaluation." In: *EURASIP Journal on Wireless Communications and Networking* 2014.1 (2014), pp. 1–15.

[71] Antonella Molinaro and Claudia Campolo. "5G for V2X Communications." In: *5G Italy White eBook* (2019).

[72] Adrian Mouat. *Using Docker: Developing and Deploying Software with Containers*. " O'Reilly Media, Inc.", 2015.

[73] *Network Emulation (netem) Linux tool*. https://wiki.linuxfoundation.org/networking/netem. Visited on 2020-11-24.

[74] Navid Nikaein, Xenofon Vasilakos, and Anta Huang. "LL-MEC: Enabling Low Latency Edge Applications." In: *Proc. IEEE CloudNet*. 2018.

[75] Yong Niu et al. "A survey of millimeter wave communications (mmWave) for 5G: opportunities and challenges." In: *Wireless networks* 21.8 (2015), pp. 2657–2676.

[76] *OMNeT++, Discrete Event Simulator*. Visited on 2020-11-20. URL: https://omnetpp.org/.

[77] *Open Source MANO (OSM)*. Visited on 2020-10-30. URL: https://osm.etsi.org/.

[78] *OpenAirInterface - 5G software alliance for democratising wireless innovation*. Visited on 2020-11-05. URL: https://www.openairinterface.org/.

[79] *OpenAirInterface, 5G software alliance for democratising wireless innovation*. http://www.openairinterface.org. Visited on 2020-11-22.

[80] *OpenStack Devstack*. https://docs.openstack.org/devstack/latest/. Visited on 2020-11-25.

[81] *Openstack Webpage*. https://www.openstack.org/. [Online; accessed 2020-11-22]. 2019.

[82] Crash Avoidance Metrics Partnership and Vehicle Safety Communications Consortium. *Vehicle safety communications project: Task 3 final report: Identify intelligent vehicle safety applications enabled by DSRC*. Tech. rep. Nat. Highway Traffic Safety Admin., U.S. Dept. Transp., Washington, DC, USA, Rep. DOT HS 809 859, 2005.

[83] *Ppen source ASN1 compiler*. http://lionet.info/asn1c/compiler.html. Visited on 2020-11-24.

[84] *Prometheus*. https://prometheus.io/. Visited on 2020-11-26.

[85] 5GPPP Programme Management Report. *5G PPP Phase II KPIs (Annex)*. Tech. rep. 2018.

[86] Z Riaz, DJ Edwards, and A Thorpe. "SightSafety: A hybrid information and communication technology system for reducing vehicle/pedestrian collisions." In: *Automation in construction* 15.6 (2006), pp. 719–728.

[87] *Ryu SDN Controller*. https://ryu-sdn.org/. Visited on 2020-11-25.

[88] Eryk Schiller et al. "CDS-MEC: NFV/SDN-based Application Management for MEC in 5G Systems." In: *Computer Networks* 135 (2018), pp. 96–107.

[89] Peter Schmitt, Bruno Landais, and Frank Yong Yang. *Control and User Plane Separation of EPC nodes (CUPS)*. Tech. rep. 3GPP, July 2018. URL: http://www.3gpp.org/cups.

[90] *SIMUlte: LTE User Plane Simulation Model for INET & OMNeT++*. Visited on 2020-11-04. URL: https://simulte.com/add_veins.html/.

[91] Christoph Sommer, Reinhard German, and Falko Dressler. "Bidirectionally Coupled Network and Road Traffic Simulation for Improved IVC Analysis." In: *IEEE Transactions on Mobile Computing (TMC)* 10.1 (Jan. 2011), pp. 3–15. DOI: 10.1109/TMC.2010.133.

[92] Christoph Sommer et al. "How shadowing hurts vehicular communications and how dynamic beaconing can help." In: *IEEE Transactions on Mobile Computing* 14.7 (2014), pp. 1411–1421.

[93] Tejas Subramanya, Giovanni Baggio, and Roberto Riggio. "lightMEC: A Vendor-Agnostic Platform for Multi-access Edge Computing." In: *Proc. 14th International Conference on Network and Service Management (CNSM '18)*. 2018.

[94] Heikki Summala. "Brake reaction times and driver behavior analysis." In: *Transportation Human Factors* 2.3 (2000), pp. 217–226.

[95] *SUMO - Simulation of Urban MObility*. Visited on 2020-11-20. URL: https://www.eclipse.org/sumo/.

[96] Tarik Taleb et al. "On Multi-Access Edge Computing: A Survey of the Emerging 5G Network Edge Cloud Architecture and Orchestration." In: *IEEE Communications Surveys and Tutorials* 19.3 (2017), pp. 1657–1681.

[97] IT Union. "Imt traffic estimates for the years 2020 to 2030." In: *Report ITU* (2015), pp. 2370–.

[98] Vutha Va et al. "Beam design for beam switching based millimeter wave vehicle-to-infrastructure communications." In: *2016 IEEE International Conference on Communications (ICC)*. IEEE. 2016, pp. 1–6.

[99] *Veins framework*. Visited on 2020-11-21. URL: https://veins.car2x.org/.

[100] A. Virdis, G. Stea, and G. Nardini. "SimuLTE - A modular system-level simulator for LTE/LTE-A networks based on OMNeT++." In: *2014 4th International Conference On Simulation And Modeling Methodologies, Technologies And Applications (SIMULTECH)*. 2014, pp. 59–70. DOI: 10.5220/0005040000590070.

[101] Antonio Virdis, Giovanni Nardini, and Giovanni Stea. "Cellular-Networks Simulation Using SimuLTE." In: *Recent Advances in Network Simulation*. Springer, 2019, pp. 183–214.

[102] Antonio Virdis, Giovanni Stea, and Giovanni Nardini. "Simulating lte/lte-advanced networks with simulte." In: *Simulation and Modeling Methodologies, Technologies and Applications*. Springer, 2015, pp. 83–105.

[103] *VirtualBox*. Visited on 2020-11-09. URL: https://www.virtualbox.org/.

[104] *VLC media player*. Visited on 2020-11-12. URL: https://www.videolan.org/.

[105] *VMware*. Visited on 2020-11-09. URL: https://www.vmware.com/.

[106] *VMware ESXi: The Purpose-Built Bare Metal Hypervisor*. Visited on 2020-11-09. URL: https://www.vmware.com/products/esxi-and-esx.html.

[107] John Paul Walters et al. "A comparison of virtualization technologies for HPC." In: *22nd International Conference on Advanced Information Networking and Applications (aina 2008)*. IEEE. 2008, pp. 861–868.

[108] Jules White et al. "Wreckwatch: Automatic traffic accident detection and notification with smartphones." In: *Mobile Networks and Applications* 16.3 (2011), pp. 285–303.

[109] Miguel Gomes Xavier, Marcelo Veiga Neves, and Cesar Augusto Fonticielha De Rose. "A performance comparison of container-based virtualization systems for mapreduce clusters." In: *2014 22nd Euromicro International Conference on Parallel, Distributed, and Network-Based Processing*. IEEE. 2014, pp. 299–306.

[110] *Xen project*. Visited on 2020-11-09. URL: https://xenproject.org/.

This Ph.D. thesis has been typeset by means of the TEX-system facilities. The typesetting engine was LuaLATEX. The document class was `toptesi`, by Claudio Beccari, with option `tipotesi=scudo`. This class is available in every up-to-date and complete TEX-system installation.