Master Thesis

Czech
Technical
University
in Prague

F3

Faculty of Electrical Engineering
Department of Telecommunication

Offloading the Computation for Autonomous Vehicle to the Edge of Mobile Network

Bc. Anastas Nikolov

Supervisor: prof. Ing. Zdeněk Bečvář, PhD. Ing. Jan Plachý, PhD.
May 2024

MASTER‘S THESIS ASSIGNMENT

I. Personal and study details

Student's name:	**Nikolov Anastas**	Personal ID number:	**492112**
Faculty / Institute:	**Faculty of Electrical Engineering**		
Department / Institute:	**Department of Radioelectronics**		
Study program:	**Electronics and Communications**		
Specialisation:	**Technology of the Internet of Things**		

II. Master’s thesis details

Master’s thesis title in English:

Offloading the Computation for Autonomous Vehicle to the Edge of Mobile Network

Master’s thesis title in Czech:

Zpracování výpočtů pro autonomní vozidlo na hraně mobilní sítě

Guidelines:

Study principles of multi-access edge computing (MEC) for autonomous vehicles and principles of offloading decision for autonomous driving applications. Propose an algorithm for the offloading decision selecting a place of computing (locally or in MEC servers) considering latency and reliability of the task processing. Consider, for example, requirements on computing resources, quality and availability of communication resources, and availability of computing resources. Implement the proposed algorithm to a testbed of the mobile network with a model of vehicle and test effectiveness of the algorithm. Evaluate possible savings in the computing power required in the vehicle and reliability and quality of the autonomous driving.

Bibliography / sources:

[1] P. Mach and Z. Becvar, “Mobile Edge Computing: A Survey on Architecture and Computation Offloading,” IEEE Communications Surveys & Tutorials, volume 19, no. 3, 2017.
[2] T. Verschoor, V. Charpentier, N. Slamnik-Krijestorac and J. Marquez-Barja, "The testing framework for Vehicular Edge Computing and Communications on the Smart Highway," IEEE CCNC, 2023.
[3] M. S. Bute, P. Fan, G. Liu, F. Abbas and Z. Ding, "A Collaborative Task Offloading Scheme in Vehicular Edge Computing," IEEE VTC2021-Spring, 2021.

Name and workplace of master’s thesis supervisor:

prof. Ing. Zdeněk Bečvář, Ph.D. Department of Telecommunications Engineering FEE

Name and workplace of second master’s thesis supervisor or consultant:

Ing. Jan Plachý, Ph.D. Department of Telecommunications Engineering FEE

Date of master’s thesis assignment: **16.02.2024** Deadline for master's thesis submission: **24.05.2024**

Assignment valid until: **21.09.2025**

prof. Ing. Zdeněk Bečvář, Ph.D.	doc. Ing. Stanislav Vítek, Ph.D.	prof. Mgr. Petr Páta, Ph.D.
Supervisor’s signature	Head of department’s signature	Dean’s signature

III. Assignment receipt

The student acknowledges that the master’s thesis is an individual work. The student must produce his thesis without the assistance of others, with the exception of provided consultations. Within the master’s thesis, the author must state the names of consultants and include a list of references.

.

Date of assignment receipt | Student’s signature

Acknowledgements

I would like to express my deepest gratitude to my supervisors, prof. Ing. Zdeněk Bečvář, Ph.D., and Ing. Jan Plachý, Ph.D., for their invaluable guidance, support, and encouragement throughout the course of my research. Their expertise and insights were instrumental in shaping this work.

Special thanks to my colleagues and friends who provided moral support and constructive feedback, helping me to refine my ideas and approaches.

I would also like to extend a heartfelt thank you to Taylor Alison Swift, whose music provided me with inspiration and motivation during the long hours of research and writing.

Finally, I would like to extend my heartfelt appreciation to my family for their unwavering support, patience, and understanding throughout this journey. Without their encouragement and belief in my abilities, this thesis would not have been possible.

Declaration

Abstract

The advancement of autonomous vehicles (AVs) capabilities necessitates substantial computational power to process extensive sensor data and execute complex algorithms in real-time. This thesis explores the application of Multi-access Edge Computing (MEC) to offload computational tasks from AVs to MEC servers in a mobile network, thus enhancing AV capabilities in terms of computation power, while reducing onboard computational load. The primary objective of the thesis is to develop and validate an algorithm for offloading decision that considers total (communication plus computing) latency, communication resources and task computation requirements. The proposed solution encompasses a novel MEC architecture designed for scalability and flexibility of deployed services, capable of supporting numerous AVs with diverse computational requirements. The proposed architecture includes a dynamic management system of computational resources to handle fluctuating quality of the communication channel and load of the MEC server. In the proposed architecture a Digital Twin (DT) of AV is created in the MEC to enable selective computation offloading of AV processes and providing a synchronized digital replica capable of optimizing AV functionalities. This setup ensures that offloaded tasks operate in real-time conditions identical to the physical AVs. Key contributions of this thesis include the development of a robust task offloading algorithm that effectively manages the task offloading under varying mobile network conditions. The algorithm dynamically decides whether to process tasks locally or offload it to the MEC server. The offloading decision is based on parameters such as task deadlines, task size, and communication channel state. The effectiveness of the proposed system is validated through extensive testing on a real AV model offloading the tasks from the AV to the real MEC server over 5G mobile network, demonstrating significant improvements in energy consumption (up to 58 % energy saved), and the ability to meet real-time processing deadlines (60 - 90 % increase in ratio of tasks processed within required deadline).

Keywords: Multi-access Edge Computing, Computation Offloading, Offloading Decision, Real-Time, Autonomous Vehicles

Abstrakt

Pokrok v oblasti autonomních vozidel (AV) vyžaduje značný výpočetní výkon pro zpracování dat ze senzorů a provádění složitých algoritmů v reálném čase. Tato diplomová práce zkoumá aplikaci Multi-access Edge Computing (MEC) pro zpracování výpočetních úloh z AV na MEC servery v rámci mobilní sítě, čímž se zvýší výpočetní výkon AV a sníží se zatížení výpočetních prostředků ve vozidle. Hlavním cílem této práce je vyvinout a ověřit algoritmus pro rozhodování o přesunu výpočtů, který zohledňuje celkovou (komunikační a výpočetní) latenci, komunikační zdroje a požadavky na výpočetní úlohy. Navrhované řešení zahrnuje novou MEC architekturu navrženou pro škálovatelnost a flexibilitu nasazených služeb, schopnou podporovat množství AV s různými výpočetními požadavky. Navrhovaná architektura obsahuje dynamický systém řízení výpočetních zdrojů, který se vyrovnává s kolísající kvalitou komunikačního kanálu a zatížením MEC serveru. V navrhované architektuře je vytvořeno digitální dvojče (DT) AV v MEC, který umožňuje selektivní přenos výpočetních procesů AV a poskytuje synchronizovanou digitální repliku schopnou optimalizovat funkce AV. Tento systém zajišťuje, že přenesené úlohy fungují v reálných podmínkách identických s fyzickými AV. Hlavními přínosy této práce je vývoj robustního algoritmu pro přesun úloh. Algoritmus dynamicky rozhoduje, zda úlohy zpracovat lokálně nebo je přenést na MEC server. Rozhodnutí o přenosu je založeno na parametrech, jako jsou termíny úloh, velikost úloh a stav komunikačního kanálu. Efektivita navrhovaného systému je ověřena rozsáhlým testováním na skutečném modelu AV, který přenáší úlohy na reálný MEC server přes 5G mobilní síť, což prokazuje významné zlepšení spotřeby energie (úspora energie až 58 %) a schopnost splnit termíny zpracování v reálném čase (60 - 90 % nárůst poměru úloh zpracovaných ve stanoveném termínu).

Klíčová slova: Multi-access Edge Computing, Přesun Výpočetních Úloh, Rozhodování o Přesunu Úloh, Zpracování v Reálném Čase, Autonomní Vozidla

Contents

Figures

Tables

Chapter 1

Introduction

As mobile devices like smartphones and laptops have evolved, they have enabled a plethora of new applications. These devices, however, face significant limitations due to their battery lifetime and the computing power required to run complex applications. Traditionally, to overcome these limitations, tasks requiring substantial computing power can be transferred to a centralized cloud, i.e., large data centers that can handle processing computation demanding tasks far from the user [1]. This process, known as "computation offloading," helps to preserve the device's battery and allows to run applications beyond the device's inherent processing capabilities. While offloading to a centralized cloud can be beneficial, it introduces a challenge in the form of communication delay. When data is exchanged between device and the centralized cloud, it is transported over significant distances, hence, resulting delays are not suitable for real-time applications. For applications that need instant responses (real-time gaming, hard deadline processes), these delays render offloading infeasible. This limitation necessitates a solution that can handle complex computations without the associated delay [2].

Multi-Access Edge Computing (MEC) presents a feasible solution to strict latency requirements, moving away from the centralized model of cloud computing towards a distributed approach. The core concept of MEC involves bringing computational power closer to the end users by embedding these resources directly at the network's edge, such as at gNodeB (gNB). This proximity reduces latency, making it suitable for applications that demand quick response times and high computational loads. MEC is particularly beneficial for supporting a variety of computation-intensive and latency-sensitive applications on resource-limited mobile devices. By minimizing the distance that user's data travel for processing, MEC effectively reduces both the communication latency and the energy consumption associated with computing. These advantages are crucial for fulfilling the demanding requirements of the new era Internet technologies like the Internet of Things (IoT) and immersive digital experiences. Furthermore, the deployment of MEC is aimed at merging wireless communication with mobile computing. This merger has led to innovative designs in computation offloading and network architecture, which are vital for enabling new services that were previously unfeasible due to latency issues. For instance, virtual reality and complex industrial applications can now benefit from the reduced latency and increased processing power provided at the edge [3].

The growing popularity of IoT is rapidly increases the number of devices connected to the internet, such as smart home appliances, health trackers or entertainment devices [4]. One of the emerging technologies are autonomous vehicles (AVs). AVs bring enhanced safety, efficiency, and sustainability as the automotive industry evolves [5]. These AVs use artificial intelligence and cutting-edge sensors to navigate roadways with limited amount of human assistance [6]. AVs should be able to handle massive amount of data in real-time in order to make decisions that are dependable and made right away. The decentralized nature of MEC allows for the processing of applications, which require low latency, thus releasing computational resources to localized critical functions [7]. This ensures that the most crucial decisions, which require instantaneous processing to maintain safety standards, are handled locally without delays. Other less critical functionalities can benefit from the broader computational resources offered by MEC, enhancing the vehicle's capabilities without compromising core operational safety. The problematic nature of processing safety-critical tasks at the edge of the network is described in [8]. Considering traveling speed of vehicles even in metropolitan areas, the AV travels several to dozens of meters before receiving results. The delay between transmission of input data and reception of processed offloaded data should be low enough so that the AV has enough time left for reaction. The delay increases with the number of vehicles in the proximity of a MEC server which has to distribute the computation resources among the AVs.

Autonomous driving systems require substantial computing power to process extensive sensor data. To mitigate this, a hybrid approach using vehicular cloud computing and edge computing is proposed in [9] to distribute computational tasks among vehicles, roadside infrastructure, and cloud servers, thereby enhancing response time, reliability, and bandwidth usage. However, challenges remain in efficiently offloading tasks to edge or cloud while managing communication bandwidth fluctuations and ensuring timely data processing, necessitating an advanced method to optimize quality of service through elastic task offloading. This thesis proposes a solution for effective task offloading as well as architecture for flexible MEC system capable of serving large number of AVs.

Progress in simulation technologies enables the possibility to create detailed digital twin (DT) of most devices, including AVs. DT technology is an innovative solution for AV optimization. Many features of an electric AV can be computerized to increase its efficiency, performance, and smartness [10]. DT hosted on a MEC server can be updated in real-time, ensuring that offloaded applications are run in the same conditions as in the real AV. In this thesis, a DT of a model AV for experimental purposes is created and all offloaded applications that are part of the AV's operating system are part of this DT to ensure stable environment that behaves predictably with respect to the AV.

Unlike mobile phones, which are commonly associated with application offloading, AVs are highly mobile and experience a significant number of handovers [11]. Most applications that make predictions rely on a system model whose state is iteratively updated and corrected; without this, making continuous and precise predictions would be impossible. Therefore, it is crucial to maintain the state of the offloaded application during handovers. A solution proposed in [12] utilizes the Camino [13] framework to forward messages, such as those from Advanced Driver Assistance Systems (ADAS), over vehicular communication technologies. Another closely related challenge is service scaling. Prior to a handover, the environment on the next MEC server should be prepared for the incoming AV. This thesis proposes a solution for horizontal scaling, which allows a single MEC server to serve multiple AVs.

Application execution framework for MEC-based applications is proposed in [14]. The proposal requires that every gNB has its own MEC server. The MEC server collects information about communication and computational resources. The MEC system proposed in this thesis enables connection of a single MEC server to multiple gNBs. Thus, the rest of the MEC servers are able to use 100 % of computational resources for the offloaded applications. DT-assisted framework is introduced in [15]. This framework utilizes DT of the whole network as well as DTs of connected vehicles. Individual DTs of the vehicles can interact with each other over the internet (inter-twin communication) for real-time data alignment. The edge layer of the system contains a centralized controller equipped with DT powered by a deep reinforcement learning (DRL) model. The system proposed in this thesis also utilizes DTs of connected vehicles, however, there is no need for complex DT of the entire network as we do not use a centralized controller depending on DRL. This fairly simplifies both design and implementation of the architecture. Finally, [16] proposes a solution for resource management in multi-access edge computing networks. This solution considers both cooperative and non-cooperative vehicles. Each MEC server has a DRL powered controller installed. The proposed controller manages computational, communication and storage resources. Note that all of these works present only results of numeric simulations. The solution proposed in this thesis is deployed on real hardware and presented results are based on data from operational AV.

In big cities, traffic density can be extremely high, often resulting in congested networks that challenge communication resource availability for AVs [17]. Conversely, in less populated areas, the sparse traffic is usually outweighed by poor network coverage. These conditions can prevent AVs from having sufficient resources to offload all tasks they would offload under optimal conditions. To address this, an algorithm is designed to decide which tasks should be offloaded and which should be processed locally. As outlined in [18], AVs can potentially leverage computational resources from nearby vehicles through a short-range interface dedicated to vehicle-to-vehicle (V2V) communications.

Over the recent years, many algorithms for scheduling task offloading have been proposed. Early ones consider two-level offloading systems [19], which is not relevant for this thesis because we assume MEC servers with enough computation power to host virtual environments of AVs without support from centralized cloud. In contrast, very recent publications, such as [20], have embraced machine learning, proposing complex solutions that employ deep learning and Q-agents. However, these are unnecessarily complex for target scenario of this thesis, which involves AVs with very limited computational resources that need to offload as many tasks as possible, some with varying degrees of dynamic priority and strict deadlines. Some of the tasks cannot be run locally at all (non-safety critical), while others, when executed locally, should resort to simpler algorithms, resulting in limited precision. Conversely, many proposed algorithms assume that vehicles have sufficient resources to run tasks locally [21], or sometimes have idle resources that can be leveraged through collaborative offloading [7][21].

The offloading decision algorithm requires several parameters to make informed decisions. Priorities serve as one of the most important parameters considered by the algorithm. When faced with limited computation resources to the point where it is not possible to run all the applications locally, priorities signal to the algorithm which applications the AV is most desperate to offload. This becomes increasingly important in situations where communication resources are scarce, and it is not possible to offload every process the AV requires.

Another crucial parameter is the deadline of individual tasks. Combined with the size of the input/output data, state of the communication channel and availability of communication resources, the algorithm estimates if delay requirements for the processed task can be met. Leveraging the architecture of the MEC, the algorithm can be scaled easily. This is achieved by incredibly low resource consumption, each instance of the algorithm is intended to process requests from a single AV. This design imposes virtually no restrictions on the flow of requests from the AV to the MEC manager. Thus, the requests can be periodic or asynchronous, and AVs can dynamically switch between these approaches depending on their requirements.

This thesis makes following contributions to the field of MEC and its application to AVs. First, it introduces a novel architecture for a flexible and scalable MEC system[1] capable of supporting a large number of AVs with diverse computational needs. This architecture includes a dynamic manage-

[1]The code is open-source and accessible from `https://gitlab.fel.cvut.cz/mobile-and-wireless/autonomous-driving`

ment system designed to handle fluctuating network load. Second, the thesis presents a robust task offloading algorithm designed to prioritize and manage computational tasks effectively under varying conditions of network congestion and resource availability. This algorithm is tailored to handle the unique challenges posed by the real-time processing requirements of AVs. Additionally, the thesis leverages DT technology to create a virtual replica of AVs, which simplifies deployment and provides stable environment for offloaded applications. The proposed solutions are validated through implementation on real hardware, providing empirical evidence of their effectiveness and feasibility in practical scenarios. Through experiments, we demonstrate that solution proposed in this thesis is not only feasible for deployment on a real AV, it increases AV's performance and is able to operate in adverse network conditions where full offloading without any management fails. During the development of the offloading architecture and algorithm, it became clear that quality of the autonomous driving is not a suitable metric, as similar performance is observed for local processing and offloading. However, significant difference is seen in the total delay and consumed energy. Therefore, after consulting the supervisors of this thesis it was decided to show the energy savings instead.

The remainder of this thesis is organized as follows: Chapter 2 highlights existing approaches and identifies gaps that this thesis aims to address. Chapter 3 details the proposed MEC architecture, including the design and functionalities of the dynamic management system and the integration of DT technology. Chapter 4 describes the implementation of proposed architecture. Chapter 5 presents the testing methods used to validate the proposed solutions. Chapter 6 analyzes the experimental results, comparing the performance of the proposed system with existing methods and discussing the implications of the findings. Finally, Chapter 7 concludes the thesis with a summary of contributions, potential applications of the research, and suggestions for future work.

Chapter 2
State of the Art

This chapter delves into the state of the art in MEC architectures and offloading algorithms. We explore the gap between theoretical frameworks and their practical applications, particularly in AV systems, assessing how these innovations enhance computational efficiency. Detailed examinations of both the architectural setups and offloading mechanisms illustrate the ongoing advancements and challenges in integrating MEC into broader technological ecosystems.

2.1 MEC Architectures

In this section, we explore the cutting-edge advancements in MEC, particularly focusing on frameworks and architectures developed by the European Telecommunications Standards Institute (ETSI), Intel and open-source initiative OpenAirInterface (OAI).

2.1.1 ETSI-MEC Framework

ETSI is one of the European leading authorities standardizing telecommunication technologies. In [22], a framework is proposed, utilizing a virtualisation infrastructure on top of which the MEC applications are residing. Figure 2.1 shows the high-level architecture of the MEC framework, split into three main entities - networks, host and system levels. Similar approach is adopted by this thesis.

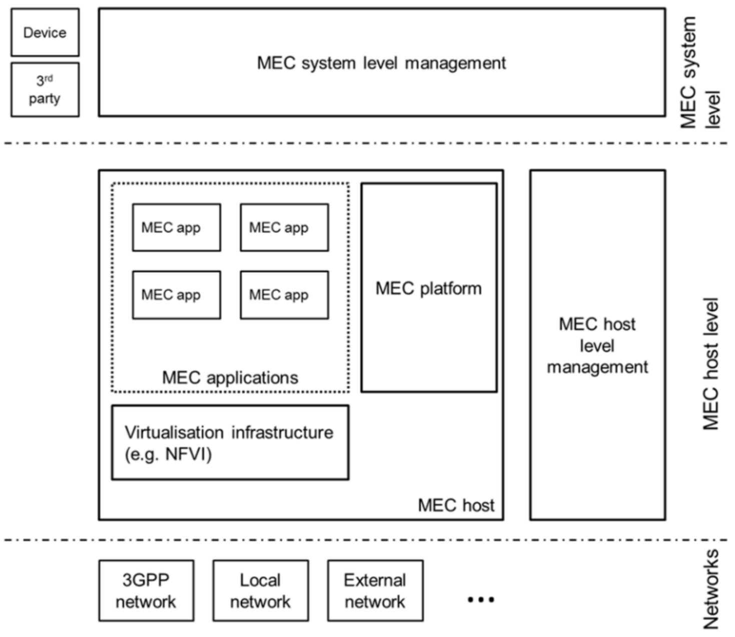


Figure 2.1: Multi-access Edge Computing framework [22].

In Figure 2.2 we see detailed architecture of the framework. The host level comprises of the MEC host itself, MEC platform manager and MEC platform.

The MEC host is providing computation, storage and network resources for the MEC applications. The MEC platform is responsible for offering an environment where the MEC applications can discover, advertise, consume and offer MEC services [22]. In case of AVs, this functionality can be replaced by DT as service discovery is handled within the car (or the DT). Thus, for MEC in context of AVs, MEC platform and MEC platform manager are redundant. The MEC applications run in a virtual environment such as a Virtual Machine (VM) or a containerised application, on top of the Virtualisation infrastructure provided be the MEC host [22]. When offloading AVs applications, there are several options how to virtualize the applications. A full-scale DT is not the only possibility to move AVs functionalities to a virtual environment. Containerization method described in [23] proposes a feasible solution for getting applications to the MEC host, using the virtualisation infrastructure.

The MEC system level management comprises of two main entities - MEC orchestrator and Operation Support System (OSS). The MEC orchestator is responsible for on-boarding of application packages, checking their integrity and authentication. Another important functionality of the MEC orchestrator is the selection of appropriate MEC hosts for application instantiation. However, this approach is in favor of stand-alone applications that can be independently instantiated and terminated. The role of MEC orchestrator designed for AVs offloading shifts towards service scaling. The MEC orchastrator no longer deals with individual applications but with the whole virtualized AV. The architecture should account for potentially thousands of users that the MEC framework has to handle. This calls for a distributed version of the MEC orchestrator. Each instance should handle certain amount of MEC hosts with main focus on horizontal scaling (i.e., cloning of DT, application containers or other forms of virtual environments depending on implementation) and load-balancing. The OSS receives requests via the CFS portal for instantiation and termination of applications and decides on the granting of these requests. This entity is also redundant for MEC systems handling AVs as instantiation/termination of applications is handled entirely by the distributed MEC orchestrators.

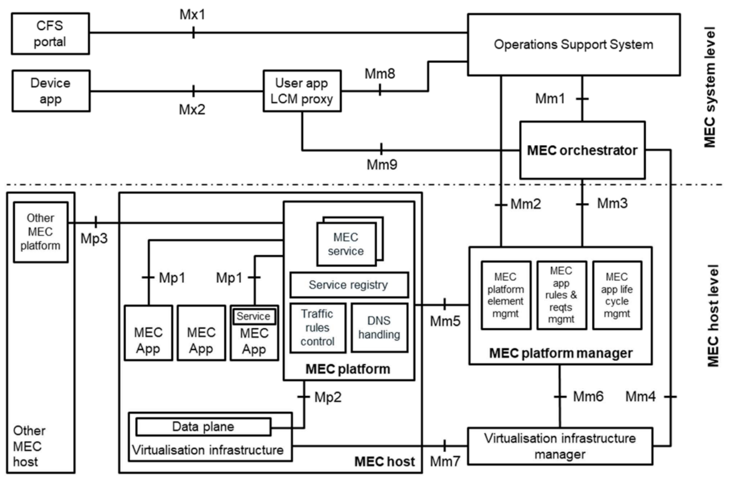


Figure 2.2: Multi-access edge system reference architecture [22].

Aside from ETSI, there are other players developing MEC solutions. Intel's Smart Edge Open toolkit [24] is a significant player in the field of MEC solutions, providing open-source tools tailored for various edge applications, including private 5G Wide Area Networks (WANs), Artificial Intelligence (AI) workloads, access edges, and Secure Access Service Edge (SASE). Despite its comprehensive nature, integrating this toolkit into the complex systems of AVs presents substantial challenges, often necessitating extensive customization to meet specific AV requirements. The toolkit's optimization for Intel hardware, such as Central Processing Units (CPUs), Field Programmable Gate Arrays (FPGAs), and GPUs, may constrain automotive manufacturers who prefer diverse or specialized automotive-grade chips, potentially limiting broader adoption due to hardware dependency. Additionally, the lack of extensive public documentation on the system's architecture complicates the evaluation of its compatibility with existing automotive technologies, possibly affecting vehicle performance and integration [24]. On the other hand, OpenAirInterface (OAI) focuses on developing a 3GPP-compliant software stack for the Radio Access Network (RAN) and the Core Network (CN) of cellular networks, supporting both 4G and 5G technologies on general-purpose computing platforms. OAI's MEC functionalities, such as the Radio Network Information Service (RNIS), enhance the adaptability of MEC applications to network conditions and simplify the orchestration, monitoring, and maintenance of RAN and CN functions through its Operations and Maintenance (OAM) project group. This open-source initiative promotes the adoption of advanced mobile network features and edge computing technologies, making it suitable for applications like autonomous driving, IoT, and industrial automation. The Mobile Edge Computing Platform (MEP) of OAI, while still under development and somewhat unstable for direct deployment, offers a scalable and maintainable architecture with integration features that align well with the needs of the automotive industry [25].

2.2 Offloading Decision Algorithms

Offloading algorithms play a pivotal role in enhancing computational efficiency. These algorithms make decisions on which tasks shall be offloaded (transferred) from resource-constrained devices to more powerful computational nodes, such as MEC servers. In this section we will go through state of the art algorithms, discussing their advantages and problems.

Algorithm proposed in [26] uses combination of multi-agent reinforcement learning (MRL) and potential game (PG) theory to make the offloading decision. The design works with a model where some vehicles have idle computational resource and other vehicles can use them for task offloading. This assumption is ambitious, considering it requires significant cooperation and compatibility among vehicles. Even in laboratory conditions where achieving necessary compatibility would be possible, not every task can be offloaded to other vehicle. The approach works well for stateless applications; however, stateful applications are unable to function without updating the model the AV operates with. Therefore, statefull applications should be processed locally or on a DT which is periodically updated. Implementing this approach for stateful applications necessitates frequent data exchanges between all AVs, rendering it impractical.

Another disadvantage is the necessity for offline training. Significant part of the algorithm is executed on the vehicle. Apart from the fact that this consumes additional, already scarce, resources of the vehicle, the design relies on integration of third party business logic into every vehicle that wants to be part of the offloading process. That is challenging from development point of view as it is virtually impossible to keep every vehicle up to date with current version of the software deployed on the network.

A digital twin-assisted algorithm is proposed in [27]. This algorithm aims to reduce the decision space by avoiding unreasonable decisions with the help of DT network. Second goal of the algorithm is to reduce overall cost of the system by employing a Deep Reninforcement Learning (DRL) algorithm to train the offloading strategy.

The system model of this algorithm envisions a network of MEC servers wirelessly connected to vehicles and wired to a high-performance cloud layer managing these servers. Vehicles share computational resources to relieve the burden on MEC servers, which host DTs of connected vehicles, integrating vehicle functionalities on the server. However, the detailed DTs, as described in [27], significantly increase storage and computation demands on MEC servers, outweighing the benefits of vehicle-to-vehicle offloading. By using only coarse DTs, saved resources can be used for MEC management, reducing latency compared to cloud layer management. As illustrated in this thesis, a coarse DT focusing on software components rather than hardware and surroundings of the AV, is sufficient and puts less strain on the MEC server compared to detailed DT. The DT network (DTN), created through wired connections between MEC servers, contains extensive vehicle and environmental data, resulting in considerable overhead. The algorithm uses the DTN to form clusters of vehicles for efficient scheduling, followed by a deep reinforcement learning (DRL) process that considers economic factors, complicating optimization by balancing vehicle performance and economic behavior. This approach imposes high resource demands on MEC servers and employs complex procedures that could be simplified, as demonstrated in this thesis.

2.3 Running Offloading Components

Any MEC framework has to utilize some sort of virtual environment where the offloading components are executed. When offloading various applications which have different requirements on the system it is often necessary to provide isolated environments. This can be achieved either by providing multiple virtual machines (VMs) or by using containers. In [28], performance of VM is compared to container-based Docker and Podman. Although VM slightly outperforms both Docker and Podman when running a single container, the performance drop is significantly in favor of container-based virtualization when multiple containers are introduced. Difference in performance between Docker and Podman is very small, letting us choose one of them based on other factors.

Crucial requirement on container-based virtualization is for it to have minimal effect on communication performance. Study [29] shows, that container-based virtualization does not introduce noticeable performance loss in communication, computing and intelligence, which indicates that containerization has a promising future in the edge-cloud computing paradigm.

Chapter 3
System Architecture

This chapter starts with a detailed description of the system model, laying the foundational framework for the subsequent architectural proposal. It then delves into the system architecture, dissecting the high-level design of its individual components to illustrate how they collaboratively function within the overall system. Finally, we provide a brief introduction to the software tools employed by the MEC architecture, highlighting their roles and significance in enhancing system performance.

3.1 System Model

Figure 3.1 depicts the system model utilized in this thesis. We assume N AVs connected to one of the M gNBs through 5G network. Each gNB has wired connection to at least one MEC server and each worker node is manged by exactly one manager node. Each manager node hosts multiple instances of the offloading algorithm depending on traffic density and the number of worker nodes managed. We assume that wired connections have negligible latency, meaning, devices with direct wired connection are in close physical proximity. In the system model, AVs are independent of each other, they share no information or resource and there are no V2V connections between them. Individual AVs have limited computational resources, and therefore exploit the computation power of MEC servers.

It is assumed that worker nodes have incomparably higher computational and storage resources than a single AV. Every AV in the network has its DT hosted by a worker node close enough to neglect delays on wired connections. This means that individual MEC server may get overwhelmed, but combined with direct neighbors, they have enough resources to host all AVs that require so.

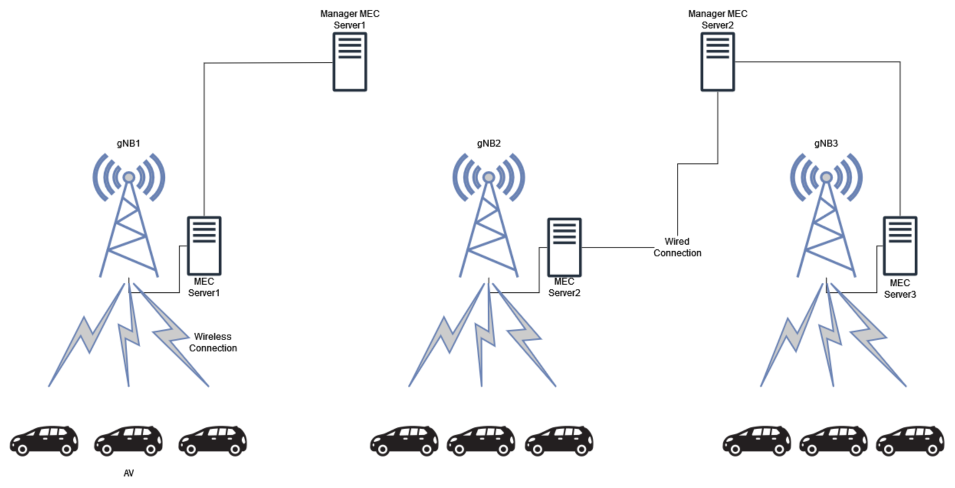
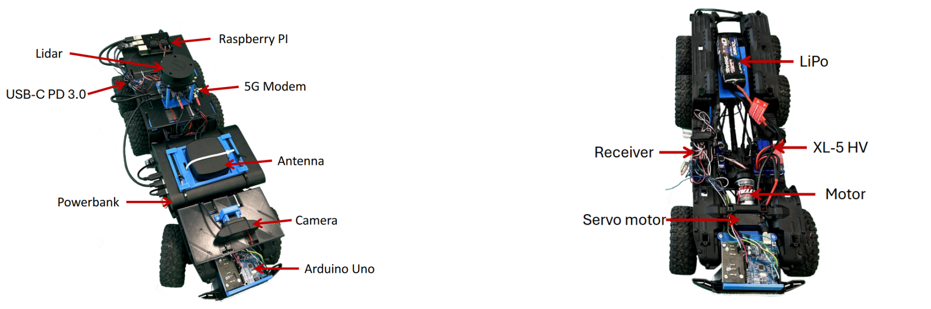


Figure 3.1: System model diagram.



(a) : AV viewed from above. **(b) :** AV viewed from below.

Figure 3.2: Photographs of the AV used for experiments.

3.2 Model of the AV

This section describes the architecture of the model AV used for experiments in this thesis. We begin by showcasing the physical construction of the vehicle, followed by the description of the software architecture and its comparison to it's DT.

In Figure 3.2 we see the hardware components deployed on the model AV used in the thesis. The main components include Light Detection and Ranging (LiDAR) for obstacle detection and map creation, camera collecting images for road sign classification convolutional neural network (CNN) and Quectel RM500Q-GL 5G modem connecting the vehicle to the network. In picture 3.2b, we see the powertrain of the AV powered by LiPo battery. Control logic for the powertrain is implemented on Arduino Uno, separating it from the main processing unit in case of critical failure.

The main processing unit of the vehicle is Raspberry Pi 4 running the

Lubuntu 20.04 operating system. The software architecture is implemented in Robot Operating System (ROS) as shown by Figure 3.3. The architecture comprises of multiple nodes, communicating via multi-cast. The ROS Master orchestrates naming and registration of services provided by individual nodes [30]. The control node serves mostly a diagnostic purpose, monitoring the state of other nodes with the ability to forcefully start or stop them. Then there are several nodes directly interacting with hardware components. These are the LiDAR node, laser scan matcher and the emergency break. The slam toolbox, position publisher, Rapidly-exploring Random Tree (RRT) and obstacle detection nodes are for path planning and navigation. Lastly, the Model Predictive Control (MPC) node optimizes vehicle control strategies in real-time.

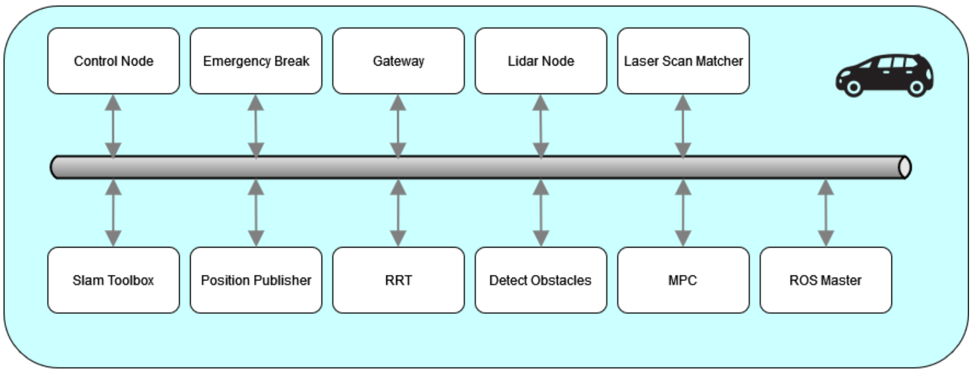


Figure 3.3: ROS architecture.

Figure 3.4 compares architectures of ROS and its DT, and how they exchange information. The dashed line between the gateways signifies that although there is no direct channel between them, there is still ongoing data exchange. It is worth noting that several nodes are missing in the DT. There is no point in creating virtual counterparts of nodes that directly interact with hardware. Slam toolbox is a software node that is not part of the DT. The purpose of the slam toolbox is to create and update a map of the AV's surroundings based on tha data from the LiDAR. If the slam toolbox would be integrated into the DT both the AV and the DT would create their version of the map with different resolution depending on the offloading process. Therefore, only the AV creates and updates the map and periodically sends it to the DT.

The gateway node plays the same role on-server and locally. Its purpose is handling of published topics, message transformation, and mainly connection to the outside world. The gateway node aggregates all messages and distributes the to their destinations. In the scenario where a node running on the MEC server needs information from a node running locally on the AV, the local gateway node sends the data to the MEC server where the DT node delegates them to the remote node. This is a key part in the offloading

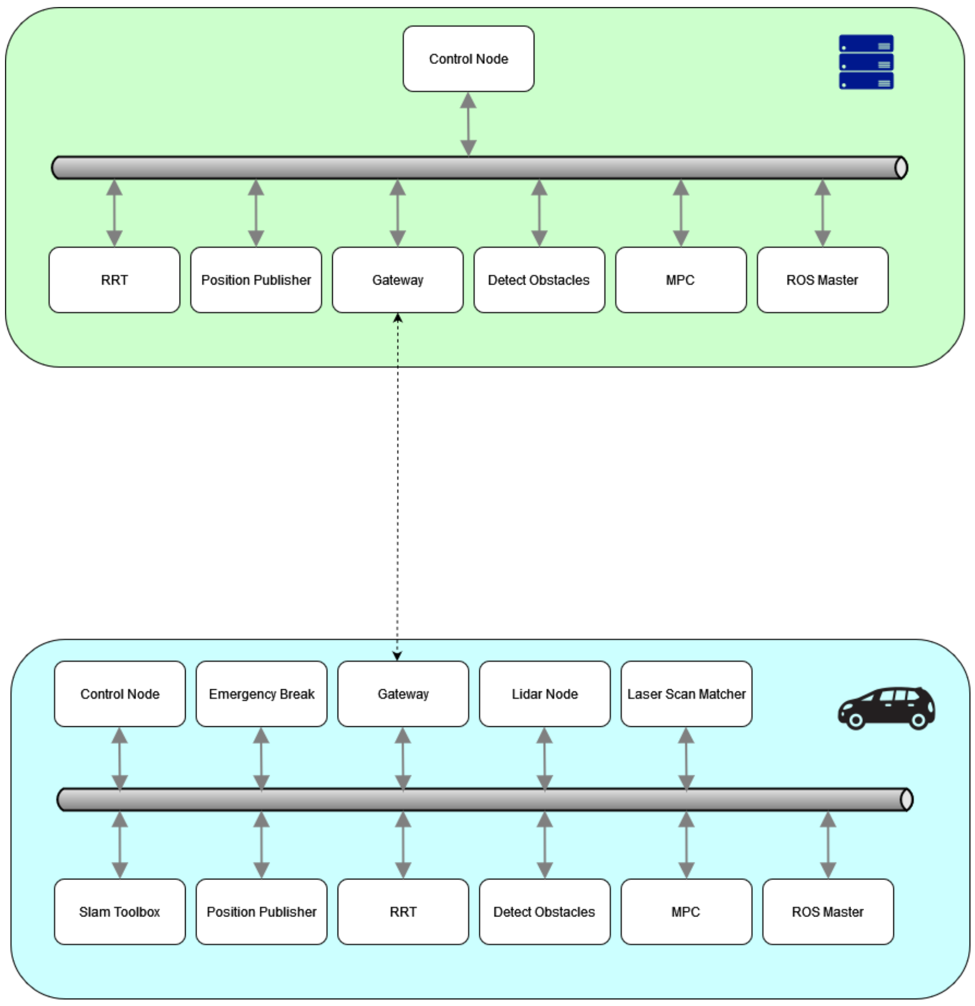


Figure 3.4: Digital Twin architecture.

procedure as it makes the communication between worker nodes and AVs manageable.

The AV implements two algorithm for path planning: RRT [31] and A* [32]. RRT quickly finds feasible paths by randomly sampling the search space and incrementally building a tree towards the goal. It excels in complex, dynamic environments where obstacles are present. On the other hand, A* is a heuristic-based algorithm that guarantees the shortest path by systematically exploring nodes based on cost. After a destination point is selected for the AV a path is determined by either of the algorithms. As discussed in Chapter 6, A* is more suitable for local execution due to its short execution time compared to RRT. Therefore, when path planning cannot be offloaded the AV uses A* while RRT is used by the MEC server to find the path for the AV.

Apart for the applications executed in the ROS environment there is another key functionality of the AV that is deployed to the proposed MEC system. It is the image processing pipeline that the AV uses to detect and classify road signs. The pipeline consists of two CNNs. First, larger network is tasked with finding the location of a road sign in an image. The second, simpler network then classifies the detected sign (e.g., speed limit, priority road etc.). Executing the image processing pipeline locally results in extreme prediction times (over 20 seconds) as the AV is not equipped with GPU making local execution impossible. This is where MEC server shows its benefit for the AV. Offloading the image processing pipeline to a MEC server with GPUs significantly reduces classification time (under 1.5 seconds) enabling the AV to detect road signs in time to adjust its driving style.

3.3 MEC Server

The MEC server is designed to host virtual environments of AVs it is currently serving. Each AV should have its own isolated environment to prevent data leakage with others, which, apart for serious privacy and security disturbance, could corrupt the applications of all affected AVs and significantly disrupt the results of stateful applications for an indeterminate period. Complete isolation between individual environments also enhances security, as authorization grants access to a single environment, protecting it from attacks or harm from other environments.

Many of the applications running on the AV do not communicate with each other or relate to one another in any way. For example, an infotainment application and an algorithm controlling the cooling system need not exchange any information; they are independent of each other and could be implemented in different programming languages, running on different operating systems. This in fact calls for further containerization at the application level, which is introduced later in this thesis. By doing so, we can not only categorize the applications by use case - control, infotainment, passenger apps, ADAS, etc. — but also provide each application with a dedicated environment for which it was developed and tested. Containerizing the applications enables us to transfer them easily while ensuring seamless integration. It allows further development of these applications from virtually anywhere without needing to access AV hardware directly.

Every worker MEC server must be able to serve multiple users connected to its gNB to minize travel distance of the data as much as possible. This necessitates the ability to scale services horizontally, another benefit of containerization scheme proposed in this work. In some cases, a worker might become overwhelmed. To address this, a load-balancing mechanism is introduced to prevent potential problems stemming from drained resources. The load-balancer should select the nearest worker with available resources to minimize data travel distance.

Every instance of this “virtual counterpart” to the AV should have a gateway application serving as an interface to the real AV. A communication gateway is designed to handle connections with AV and a manager MEC server (see 3.4). Although there would be no technical problem with direct communication between AV and manager MEC server, it is much more practical to manage everything from a single application. This approach prevents any synchronization issues; integration and diagnostics are much easier, and the external communication scheme is simplified. An equally important part is the distribution of input data to the offload application containers and the collection and forwarding of the output data back to the AV. A communication architecture addressing this is described in the detail in Chapter 4.

Aside from ROS nodes, the model AV needs to offload an image processing pipeline consisting of two CNNs for road sign detection and classification. Execution time of the pipeline on the Raspberry Pi 4 takes over 20 seconds (given it is the only running process) which makes it unusable for real life traffic, and therefore is not part of the vehicle’s architecture. On a MEC server equipped with dedicated graphics processing unit (GPU) the prediction takes less then 1.5 seconds making it a viable solution for this problem.

3.4 Manager MEC Server

Another crucial component of the MEC system is the manager MEC server. This component orchestrates the offloading process and manages scaling of worker servers. The primary task of the manager MEC server proposed by this thesis is to run the offloading decision algorithm (see 4.3), a core piece of software designed for efficient use of communication resources in the network, while taking into account the requirements of the AVs. This algorithm is computationally light, enabling the manager to scale effectively for dense traffic conditions. Such scalability is fundamental to the system’s ability to handle potentially hundreds or even thousands of AVs simultaneously, any reasonably powerful server should be able to run thousands of instances of the algorithm.

For the offloading decision to be made, the manager MEC server continuously collects and analyzes data from the Radio Access Network (RAN) of the gNB serving the processed AV. This RAN data is used to estimate the Uplink (UL) and Downlink (DL) bitrates between the AV and gNB, which are required inputs for the proposed offloading decision algorithm. The manager MEC server also plays a key role in service scaling. It monitors the performance and capacity of each worker MEC server and adjusts their workloads as necessary, spawning additional DTs for newly connected AVs if needed. If a particular server is under high load, the manager can delegate new AV to other nearby, less burdened server to ensure that service latency does not dramatically increase due to travel distance. This dynamic scaling capability ensures that the system maintains high levels of efficiency and responsiveness, even under varying and unpredictable operational conditions.

Communication between the manager MEC server and the worker MEC servers is critically important. The manager must efficiently distribute offloading decisions and updates to each AV. It also needs to handle incoming offloading requests and RAN information. The offloading requests are tailored to contain information about task deadlines, size of input/output data of the tasks required to be transmitted and task priorities. The offloading requests and offloading decisions are transmitted through the worker gateway interface. Whether the same applies to handling RAN information depends on implementation, nevertheless the source and destination points remain the same.

One of the core features of the manager MEC server is it's ability to scale horizontally depending on the number of AVs it is serving. With connection of new AV additional replica of a container executing the offloading decision process is spawned. This approach ensures that each AV has it's own independent scheduler. The lightweight nature of the container allows large number of replicas so that single manager MEC server is able to serve thousands of vehicles before it is required to involve additional server.

3.5 Software Tools Overview

This section delves into the essential software tools that make AV systems run smoothly, from deploying them in containers to processing data in real-time and using advanced communication protocols. Among these tools, Docker Tools stand out as a key player, making it easier for developers to manage multiple container applications. Docker Swarm also plays a significant role by efficiently scaling and organizing distributed environments, taking AV deployment to a new level of effectiveness and scalability. Additionally, Apache Kafka is highlighted as the platform for handling real-time data streams, ensuring seamless information exchange crucial for AV decision-making. Lastly, the integration of 5G communication marks a significant advancement, providing AV systems with unmatched bandwidth, low latency, and reliability.

3.5.1 Docker Tools

Docker-Compose is a tool for defining and running multi-container Docker applications. With Compose, a YAML file is used to configure application services, networks, and volumes. This approach simplifies the deployment, scaling, and operational tasks of container-based applications. Docker-Compose primarily targets development, testing, and staging environments, offering a straightforward way to manage complex applications as a single unit [33].

Docker-Compose comes with several features that make it particularly useful for managing multi-container setups:

- **Service Definition:** Allows defining and running multi-container Docker applications.
- **Isolation:** Each service runs in its container, ensuring isolation and reducing conflicts.
- **Scalability:** Services can be easily scaled up or down based on requirements.
- **Ease of Configuration:** Uses a simple YAML file for configuration, making it easy to define and share configurations.

Stemming from Docker-Compose's simplicity and ease of use in managing containers on a single server for a single user, Docker Swarm extends these capabilities to cater to the complexities of multi-user, multi-server systems. By harnessing Docker Swarm's scalability and streamlined management, we can seamlessly transition from a single-user, single-server setup to a distributed environment [34].

In Docker Swarm, several key concepts facilitate the management and orchestration of containerized applications across multiple nodes:

- **Service:** Defines the tasks to be executed, including the container image, ports, and replicas. Services ensure that the desired state of a containerized application is maintained within the Swarm cluster.
- **Task:** The smallest unit in Swarm, representing a single instance of a service. Tasks are scheduled and executed on nodes within the cluster based on resource availability and constraints defined by the user.
- **Node:** A physical or virtual machine within the Swarm cluster that runs Docker Engine and participates in the orchestration. Nodes can be categorized as manager nodes or worker nodes, each with distinct responsibilities in the cluster.
- **Manager Node:** Handles cluster management tasks such as scheduling, orchestration, and maintaining cluster state. Manager nodes are responsible for distributing tasks among worker nodes and ensuring the overall health and availability of the Swarm cluster.

- **Worker Node:** Executes tasks assigned by the manager nodes. Worker nodes provide the computational resources necessary to run containerized applications within the Swarm cluster, scaling horizontally to accommodate increased workload demands.

Node Management

Node management in Docker Swarm involves adding, removing, promoting, and demoting nodes within the cluster. This ensures scalability, fault tolerance, and efficient resource utilization. In context of this thesis, a Docker node is a physical server at the edge of the network. Adding nodes to the Swarm cluster increases its capacity and allows for better distribution of workload across available resources. Nodes can be dynamically added or removed to accommodate changing application requirements or infrastructure conditions. Promoting nodes to manager status increases the fault tolerance and resilience of the Swarm cluster by distributing management responsibilities across multiple nodes. Manager nodes collaborate to maintain consensus and ensure the consistency of cluster state. Demoting nodes from manager status reduces the administrative overhead and resource requirements associated with cluster management. Demoted nodes continue to participate in the execution of tasks as worker nodes while relinquishing control over cluster management operations.

Swarm Overview

To ensure isolation and security, every node deploys new set of services for each AV. These so called replicas are completely isolated, preventing mishandling of AV's data. From the AV's point of view the interaction is identical to that of single Docker-Compose deployment. After the connection is terminated, the node removes excess replicas freeing computation resources for other services.

- **Offloading Manager:** The Offloading Manager serves as the orchestrator of tasks within the Docker Swarm cluster. It runs the offloading decision algorithm and prompts individual AVs to offload tasks based on several factors such as channel quality. By intelligently distributing services to the most suitable nodes, the Offloading Manager enhances resource utilization and improves overall system performance.

- **Docker Nodes:** Docker Nodes form the backbone of the Swarm cluster and execute tasks assigned by the manager nodes. These nodes can either be assigned manager (not to confuse with offloading manager) or worker role, each contributing to the overall computational capacity and resilience of the cluster.

Manager nodes collaborate to maintain consensus and manage cluster state, while worker nodes focus on executing tasks and hosting containerized applications. The distributed nature of Docker Nodes enables horizontal scalability and fault tolerance, allowing the Swarm cluster to seamlessly adapt to changing workload demands and infrastructure conditions.

- **Docker Containers:** Docker Containers encapsulate applications and their dependencies, providing a consistent and portable environment for execution across different nodes within the Swarm cluster. Containers offer lightweight isolation, efficient resource utilization, and rapid deployment.
- **Overlay Network:** The Overlay Network in Docker Swarm facilitates seamless communication and connectivity between containers deployed across different nodes within the cluster. By abstracting network details and providing a virtualized network layer, the Overlay Network enables containers to communicate with each other regardless of their physical location or underlying network infrastructure. This allows for the creation of distributed applications that span multiple nodes, while maintaining simplicity and flexibility in network configuration.

The utilization of Docker in this thesis offers several benefits:

- **Simplified Management:** Managing a multi-container architecture becomes significantly easier with Docker-Compose.
- **Consistency:** Docker-Compose ensures consistent environments across development, testing, and production.
- **Isolation and Security:** Containers are isolated from each other, reducing the risk of conflicts and enhancing security.
- **Scalability:** Easy scaling of services to meet increased demand or processing requirements.

3.5.2 Apache Kafka

Apache Kafka is an open-source stream-processing software platform developed by the Apache Software Foundation, written in Scala and Java. Kafka functions as a distributed streaming platform that can publish, subscribe to, store, and process streams of records in real time. It is designed to handle high volumes of data, offering high throughput, built-in partitioning, replication, and inherent fault-tolerance, making it an ideal solution for large-scale message processing applications [35].

Kafka's architecture is uniquely suited for handling real-time data feeds due to its robust set of features:

- **Scalability:** Easily scales horizontally, accommodating more data and higher throughput without incurring downtime.
- **Durability and Reliability:** Ensures data is not lost and can withstand node failures.
- **Performance:** Maintains high performance even with very large volumes of data.
- **Real-Time Processing:** Capable of handling real-time data feeds with low latency.

The primary function of Apache Kafka is to build real-time streaming data pipelines and applications that adapt to the data streams. It is widely used for various purposes like tracking service calls, monitoring IoT devices, real-time analytics, and more. In the context of AVs, Kafka can serve as a powerful tool to handle the continuous stream of data generated by the vehicles. Capabilities of Apache Kafka are discussed in [36]. In this thesis Apache Kafka is utilized as a communication broker between the Docker containers on the server.

Overall, the choice of Apache Kafka is motivated by these factors:

- **High Throughput and Scalability:** Kafka's ability to handle high volumes of data and scale as needed is crucial for managing the large amounts of data transmitted between the vehicle and server.
- **Fault Tolerance:** Its distributed nature and fault tolerance ensure that the communication system is robust and can handle potential failures without data loss.
- **Real-Time Processing:** Kafka's capability for real-time data processing is essential for the immediate and synchronous communication requirements of autonomous vehicles.

Apache Kafka uses a concept called topics. Topics in Kafka are categories or feeds to which records are stored and published. They act as the channels through which data flows, allowing producers to send messages to specific topics, and consumers to subscribe and read from them. Each topic is split into partitions for scalability and fault tolerance. Brokers, on the other hand, are Kafka servers that store and manage the data for the topics. Each broker handles a portion of the partitions, ensuring high availability and load balancing across the Kafka cluster. Together, topics and brokers enable Kafka to efficiently handle large volumes of real-time data.

ZooKeeper [37] plays a crucial role in Apache Kafka by managing and coordinating the Kafka brokers. It keeps track of all the brokers in the cluster and handles leader election for partition leaders among brokers. Additionally, ZooKeeper stores configuration information for topics, quotas, and access control lists (ACLs) and tracks the state of topics, partitions, and offsets. Overall, ZooKeeper ensures the reliability and consistency of the Kafka cluster.

3.5.3 5G Communication Setup

The advent of 5G technology offers transformative potential for AV systems, providing the high bandwidth, low latency, and reliable communication necessary for their operation. This section details the setup for 5G communication between the MEC server and the AV, utilizing the N310 Universal Software Radio Peripheral (USRP) to act as a front end for running the OAI 5G RAN and 5G core network.

The 5G communication testbed is designed to closely emulate the real-world conditions that AVs will experience. The setup comprises the following key components:

- **N310 USRP and OAI 5G Platform:** The N310 USRP serves as the cornerstone of our testbed, configured together with the OAI 5G core network. This setup enables the simulation of a 5G network environment, providing a flexible and powerful platform for developing and testing 5G technologies tailored to the needs of autonomous driving.

- **Quectel RM500Q-GL Modem:** Quectel RM500Q-GL is a 5G module optimized specially for IoT/eMBB applications. Adopting the 3GPP Release 15 technology, it supports both 5G NSA and SA modes. Its advanced features include support for ultra-fast data transfer speeds, low latency, and enhanced reliability, making it ideal for applications requiring high-bandwidth data transmission, such as industrial IoT, automotive telematics, and mobile broadband. Additionally, the RM500Q-GL modem is designed with robust security features to safeguard data transmission, ensuring privacy and integrity across connected devices and networks. The RM500Q-GL modem is mounted on the AV and is configured as a dongle, connecting the AV to the 5G network.

Chapter 4

Implementation of the System Architecture

In this chapter, we discuss the practical implementation of the MEC server architecture. We will explore the DT of the AV system, including the integration of ROS nodes and the deployment of the image processing pipeline for road sign detection in a container. This chapter also covers the implementation of the communication schemes that ensure seamless interaction between the AV and the MEC server. Through detailed description of the system setup, and the challenges encountered during implementation, we aim to provide a comprehensive view of how theoretical design is transformed into a functional system that significantly enhances AV's performance.

4.1 Digital Twin of the AV

To virtualize the AV's functionalities, ROS nodes (as described in 3.2) are deployed within a single Docker container. This setup allows for scalable deployment of multiple AV functionalities across different MEC nodes. Each ROS node within the MEC infrastructure mirrors the capabilities of its real-world counterpart but operates under the computational capabilities of the edge server. By including the entire ROS node system into one Docker image we ensure that the nodes operate in the same software environment as is on the AV.

The primary challenge in implementing ROS nodes in a virtualized environment is maintaining real-time synchronization between the vehicle and the MEC server. During offloading, the nodes activated on the server are deactivated on the AV and vice versa to preserve energy. This is illustrated in Figure 4.1, where MPC and Detect Obstacles is offloaded. It is essential that the ROS container receives its messages complete and in correct order to ensure that the offloaded applications work with uncorrupted and relevant data. This is guaranteed by using Apache Kafka topics for inter-container communication, which preserve the chronological order of messages and ensure their completeness.

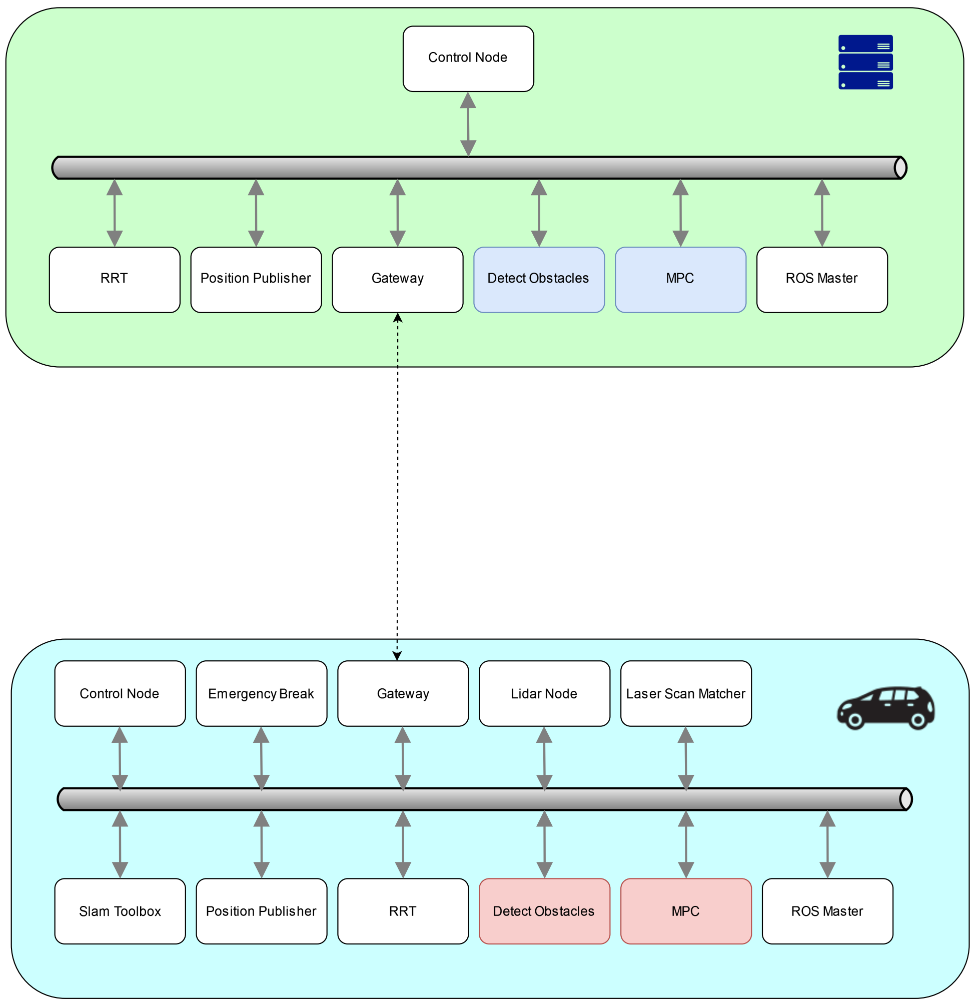


Figure 4.1: Active and inactive ROS nodes during offloading.

The ROS Control node acts as the central command center within the MEC server's ROS architecture. It is designed to manage the operation of other ROS nodes by issuing start, stop, and restart commands based on AV's operational requirements. This centralized control helps with monitoring individual nodes for testing purposes. By issuing direct Transmission Control Protocol (TCP) connection between the real and the virtual Control node (as shown in 4.4) we maintain control over the AV even in case the Gateway container crashes. The Gateway container is otherwise a signle point of connection between the AV and the MEC server. We are able to bypass this connection by opening a socket directly between the Control nodes. It is important to note that this introduces a significant vulnerability into the system's security. Therefore it is critical to remove this link after the testing phase is over.

Route planning is a fundamental component in AV operations, requiring precise map visualization to enable users to select destination points. RVIZ 2 [38], the visualization tool built into ROS serves precisely this purpose as it allows users to interact with the map and select points which the AV should achieve. RVIZ 2 collects and visualizes real-time data from the AV. Users can therefore see real-time position of the AV, path planned by the AV that leads to the selected destination and real-time LiDAR data showing obstacles dynamically (dis)appearing from the view.

The DT is periodically synchronized with its physical counterpart. Updated map and current position is sent from the real AV to the DT so that tasks that use this information can be later offloaded without the need to transmit the map, which is typically the largest object transferred to the MEC server, making the offloading of these tasks faster. However, the synchronization should not block the offloading of tasks. Therefore, the proposed offloading decision algorithm is able to postpone the synchronization should it block any tasks with higher priority than that of the synchronization process. The map is updated roughly once every 45 seconds, the frequency of the position updates is typically 2 Hz, but the size of the position message is quite small. Thus, the synchronization process causes little overhead.

4.2 Communication scheme

This section provides detailed description of implemented communication scheme between the AV, MEC server and the manager MEC server (external communication) as well as the communication between containers hosted on the MEC server (internal communication). The distinction between internal and external communication is shown by Figure 4.2 to provide more clarity. Effective communication design is crucial for reliable and efficient operation of the MEC system. Different strategies are employed for external communication with AVs and internal communication among the components of the MEC infrastructure.

4.2.1 External Communication

For external communication, the MEC system uses TCP (Transmission Control Protocol). TCP is chosen for its reliability of delivering packets in the exact order they were sent, which is crucial for maintaining the order of communication between AVs and the MEC server. TCP's error-checking features and its ability to adjust the rate of data transmission based on network conditions ensure that data transfer between the AV and the MEC server remains consistent, even over varying mobile network conditions.

The connection between the AV and the MEC server is established over a mobile network, which is assumed to provide the necessary coverage and bandwidth to accommodate AV's data traffic. The gateway container within the MEC server acts as the primary interface between the external and in-

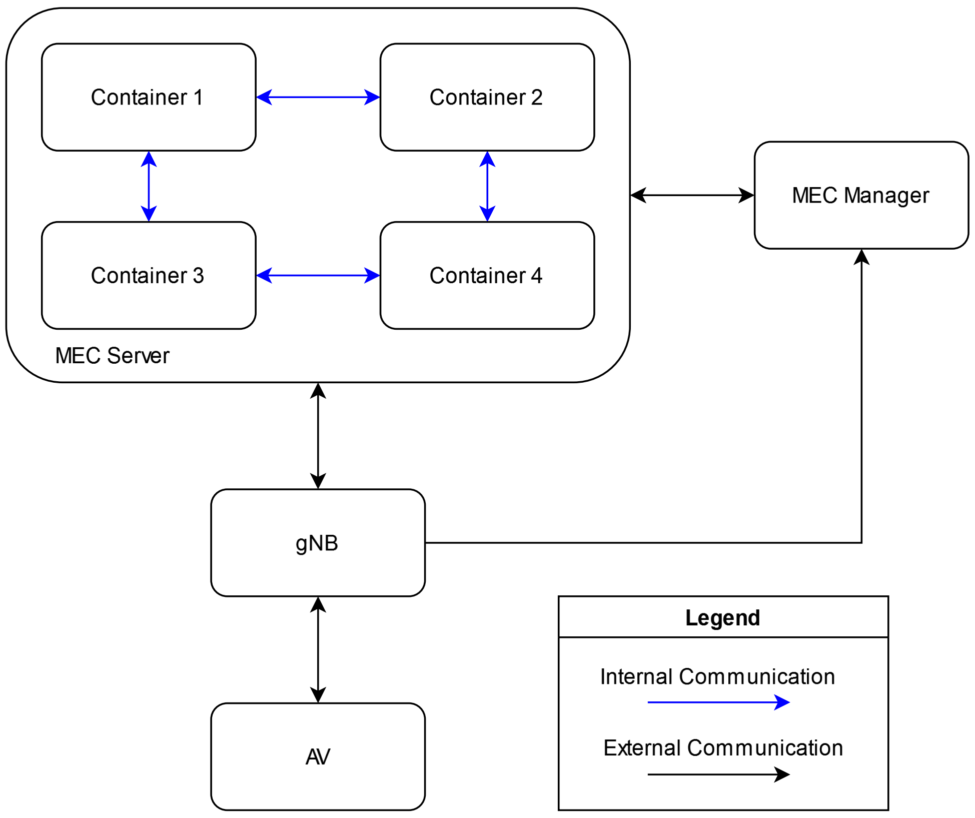


Figure 4.2: Internal and external communication between MEC components.

ternal communication, directing incoming data from AV to the appropriate containers or the MEC manager and vice versa. The gateway container mediates communication between the AV and the internal components of the MEC server. It ensures that offloading data are properly routed either to the containers or to the MEC manager for scheduling and management of tasks. This for a more organized and manageable communication flow, which is essential for maintaining system performance and responsiveness.

During the testing phase, the MEC system architecture allows for the opening of a secondary TCP channel that bypasses the Gateway container and directly connects the AV's Control node to the Control node of its DT. Figure 4.3 shows detail of the communication scheme with focus on this connection. In testing stages it is sometimes necessary to activate or kill processes on the AV. The over-the-air (OTA) diagnostics channel enables this as it sends commands directly to the AV's Control node. This channel is intended strictly for OTA diagnostics and testing purposes, providing developers with direct access to the AV without disturbing ongoing processes on the MEC server. This direct connection can be instrumental in performing real-time diagnostics and in-depth system testing, facilitating a more efficient testing of newly deployed features. It is important to note that while the secondary TCP channel is valuable for testing, the final product is designed to exclusively use the main TCP channel through the Gateway container, routing all the data through a controlled and secure gateway, minimizing potential vulnerabilities.

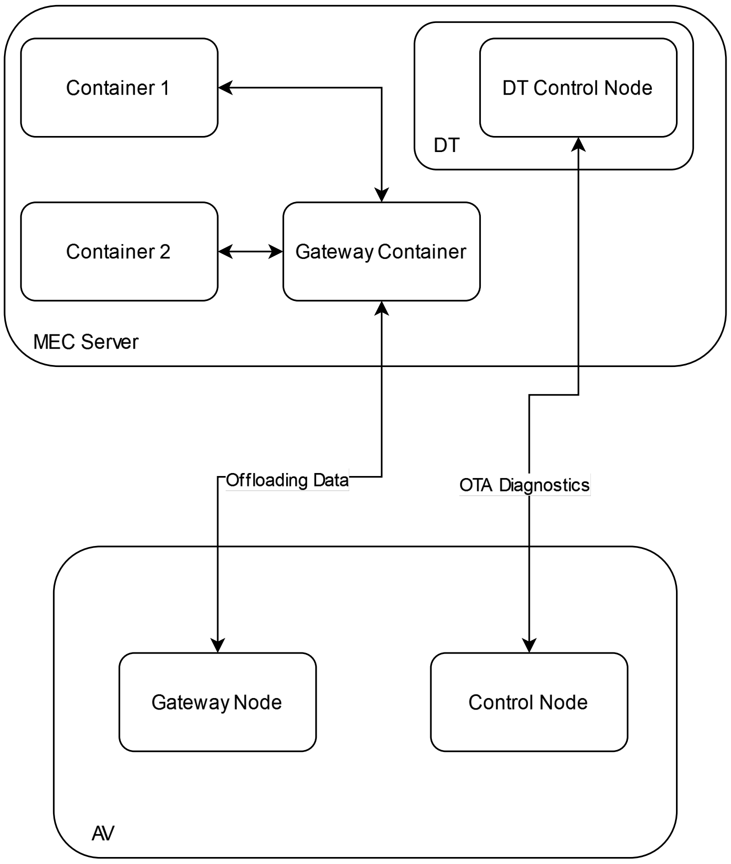


Figure 4.3: OTA diagnostics channel bypassing the Gateway container.

4.2.2 Internal Communication

Internal communication strategies for vehicle-related services often look to the state of the art in vehicle communication systems for guidance. In automotive applications, the Scalable service-Oriented MiddlewarE over IP (SOME/IP) protocol is commonly used for enabling communication between in-vehicle services. While SOME/IP is well-suited for internal vehicle communications due to its service discovery and event handling capabilities, it is out of scope of this thesis to implement said protocol to the proposed MEC system. Instead, Apache Kafka - a streaming protocol described earlier in this thesis - is used. It introduces communication flow similar to that of SOME/IP but is easier to integrate into the MEC system.

The communication between different components — namely, the manager MEC server, gateway and application containers — must be reliable and efficient in handling high volumes of data.

- **Data Streaming for Messaging:** Streaming is the continuous flow of data that is transmitted in real-time between systems, applications, or devices. It is used to send, receive, and process data in a steady, uninterrupted stream, allowing for real-time analysis, action, and interaction. A suitable approach for the realm of AVs where large amount of data is processed and exchanged between applications. For communication between the Gateway container and other containers, the data streaming approach is adopted. Streaming is facilitated through topics in a publish/subscribe model, which efficiently handles the dynamic distribution of messages based on the current operational context and the subscribing components' needs.

- **TCP for Structured Communications** For interactions transferring smaller amounts of data in more structured fashion, such as those between workers and the manager MEC server, TCP is utilized. TCP ensures that messages are delivered reliably and in order, which is crucial for control signals and task orchestration commands that require guaranteed delivery.

The combination of TCP and data streaming protocols accommodates the diverse needs of the MEC system. While TCP provides a robust and reliable channel for critical control and management communications, the use of data streaming for messaging between the gateway and application containers allows the system to maintain high throughput and flexibility. This approach ensures that the system can handle varying data loads. The communication between ROS nodes is essential for coordinating various tasks and exchanging data. Typically, ROS nodes communicate via multi-cast, enabling efficient information exchange within the AV's onboard network. However, when integrating the AV with the MEC server, a structured communication architecture needs to be implemented to facilitate seamless interaction between the vehicle and the AV environment.

Allocation of communication resources is handled entirely by the mobile network, the offloading decision algorithm only makes the offloading decisions and reservation of communication resources. Every MEC manager has access to real-time RAN information of gNBs connected to the managed worker MEC servers. The AVs are sending aperiodic offloading requests containing information about tasks they wish to offload to the worker nodes. The tasks are assigned priority based on several factors such as AVs ability to process the task locally before deadline or elapsed time since last execution. These requests are then forwarded for processing to corresponding manager node where the offloading decision is made. As it is assumed that the tasks are independent, they can be processed independently.

Implementation of the proposed communication architecture is shown in Figure 4.4. The Gateway node in the AV, establishes a TCP connection with the Gateway container in the Docker nodes. This Gateway node serves as the intermediary for collecting and transmitting offloading data to the Docker node as well as distributing data among other ROS nodes. Upon receiving offloading decision, the Gateway node forwards input data to the Docker nodes Gateway container. The input data are parsed by the Gateway container and distributed via Kafka topics to corresponding containers. The Kafka topics are hosted by a broker server image. The internal communication of the Docker node is supervised by ZooKeeper container. If the input data are intended for the ROS container, they are forwarded to the Gateway node residing inside the ROS container. Then the operation proceeds in the same fashion as onboard the AV.

When the offloadrd data are processed, the communication flow is reversed. Data originating from the server-side ROS nodes follow the multi-cast communication paradigm, reaching the ROS container's Gateway node. Subsequently, this data is transmitted to the Gateway container in the Docker nodes via Kafka topics. The Gateway container then forwards the data to the AV's ROS Gateway node, ensuring synchronized communication between the AV and its DT.

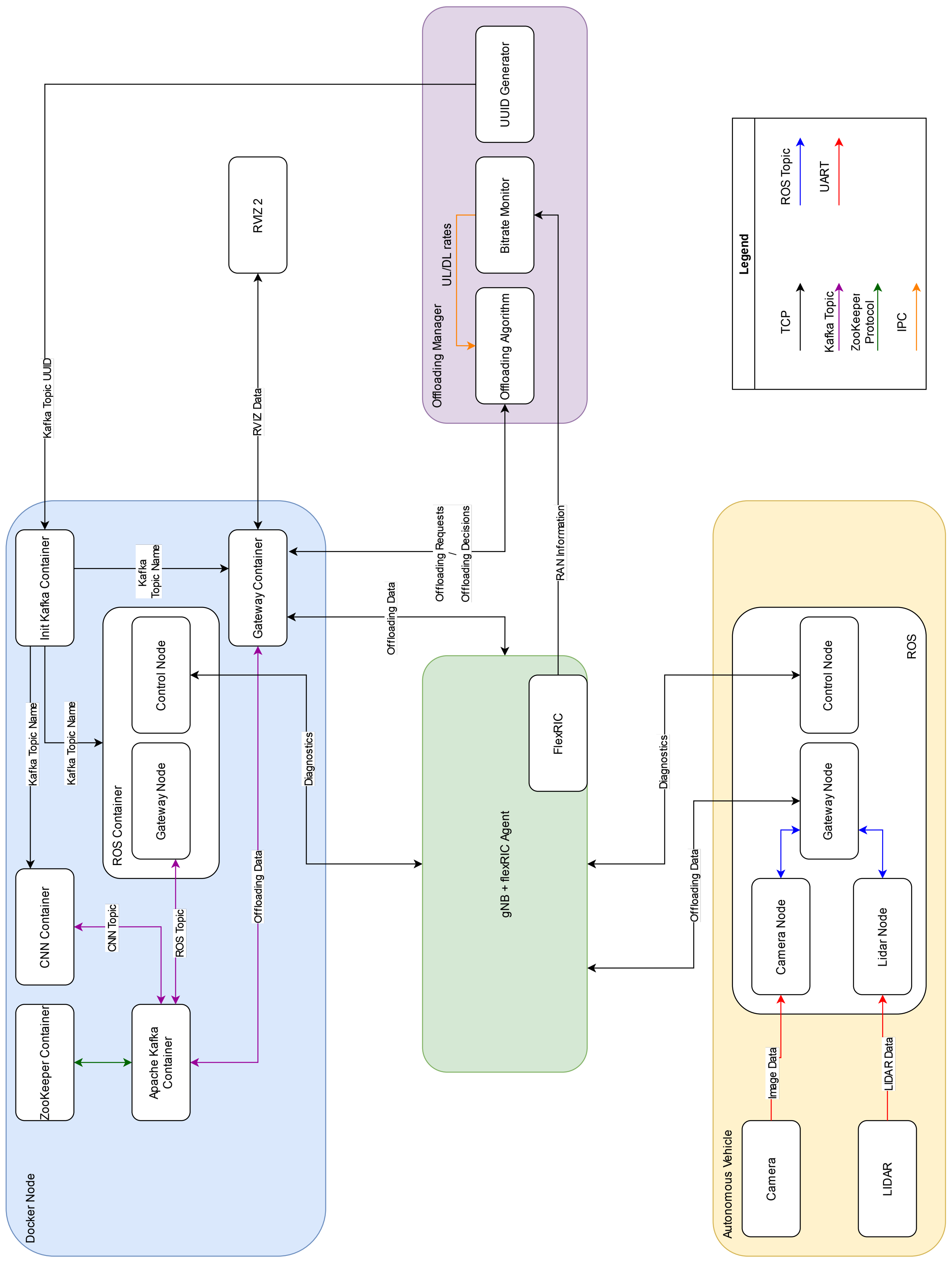


Figure 4.4: Implemented communication scheme.

When dealing with Docker Swarm (described in 3.5.1), used for horizontal scaling of the containers on the MEC server allowing to accommodate multiple AVs, integration of Apache Kafka is not as straightforward as in other cases. It is unnecessary to spawn additional replicas of the Kafka container along with replicas of other containers as one instance of Apache Kafka can handle large amounts of topics. However, it is essential for the replicas of ROS, CNN and Gateway containers to use unique topics otherwise data from one AV would get distributed to every replica. To prevent this the InitKafka container is introduced. Each time this container is replicated it requests a Universally Unique Identifier (UUID) from the MEC manager. The InitKafka container then constructs the names of individual topics using the UUID and sends them to the newly spawned containers. This way, every AV is guaranteed unique Kafka topics preventing data leakage between DTs.

4.3 Offloading Decision Algorithm

When deciding which task to offload and which task to process locally, several factors have to be considered. The main goal is to offload as many tasks possible w.r.t their priority.

The algorithm begins by receiving an offloading request containing a list of tasks from the AV. The request also contains information about task deadlines, size of data to be transmitted if the task is offloaded and the priority of the task. Tasks are assigned one of the ten priority levels (0 - highest, 9 - lowest), depending on the AV's necessity to offload them. Multiple tasks are allowed to have the same priority, in such case the algorithm favors tasks with earlier deadlines.

The tasks are sorted into priority groups (P_n, $n \in (0, 9)$). Starting with the highest priority group (P_0), a reference deadline is determined:

$$t^{\text{ref}} = \min_{i \in P_0} t_i^{\text{deadline}}, \tag{4.1}$$

where t_i^{deadline} is the deadline of task i. As the AV transmits the offloading data of all offloaded tasks at the same time, this approach ensures that offloading lower priority tasks with later deadlines does not cause higher priority tasks to miss their deadlines. Total delay of task i is estimated as

$$t_i^{\text{offload}} = \frac{8B_i^{\text{ul}}}{U} + \frac{8B_i^{\text{dl}}}{D} + 1000\frac{B_i}{f^{\text{CPU}}} \quad [\text{ms}], \tag{4.2}$$

where B^{ul} [B] is the number of bytes to be transmitted from the AV to the MEC server, B^{dl} [B] represents the number of bytes to be transmitted from the MEC server to the AV, B_i [B] is the size of task i, U [kbps] and D [kbps] are the respective data rates for uplink and downlink and f^{CPU} [Hz] is the frequency of the MEC server's CPU. The term $\frac{B_i}{f^{\text{CPU}}}$ is multiplied by

1000 for conversion from seconds to milliseconds. If the total delay exceeds the reference deadline, the task is processed locally; otherwise, the task is offloaded.

The input data for all offloaded tasks are transmitted at the same time, therefore, every time the algorithm decides to offload a task, it is accounted for calculation of next task delay. The offloading decision for each task is specified by α.

$$\alpha_i = \begin{cases} 0, & \text{local} \\ 1, & \text{offload} \end{cases} \tag{4.3}$$

The optimal offloading strategy consists of the optimal offloading decisions for requested tasks

$$\mathrm{A}^* = \bigcup_i \alpha_i^*, \tag{4.4}$$

where I is the set of tasks in the offloading request. Finding the optimal offloading strategy leads to a constrained optimization problem defined as follows:

$$\begin{aligned} & \mathrm{A}^* = \arg\max_{\alpha \in \mathcal{A}} \sum_{i \in I} \alpha_i \gamma_i \\ \text{s.t.} \quad & t_i^{\text{offload}} \leq t^{\text{ref}}, \quad \forall i \mid \alpha_i = 1 \\ & \sum_{i \in I} \alpha_i f_i \leq f^{\text{CPU}} \end{aligned} \tag{4.5}$$

We are trying to process the maximum number of tasks on the MEC server, however, the MEC server should not be overloaded by the computational strain from offloaded tasks (f_i represents computational resources occupied by task i) and the offloaded tasks should meet their deadline (t_i^{offlaod} is the total delay of the offloaded task). The heuristic value γ is introduced to up the value of tasks with higher priority. If two or more tasks have the same priority, the task with earlier deadline is assigned more value. The decision space $\mathcal{A} = \{0, 1\}$ represents possible locations for tasks processing: 0 - local, 1 - MEC server.

The formulated problem is solved by the proposed algorithm illustrated by the flow chart in Figure 4.5. After all tasks are processed, the constructed decision is transmitted to the Gateway container from which it is forwarded to the AV.

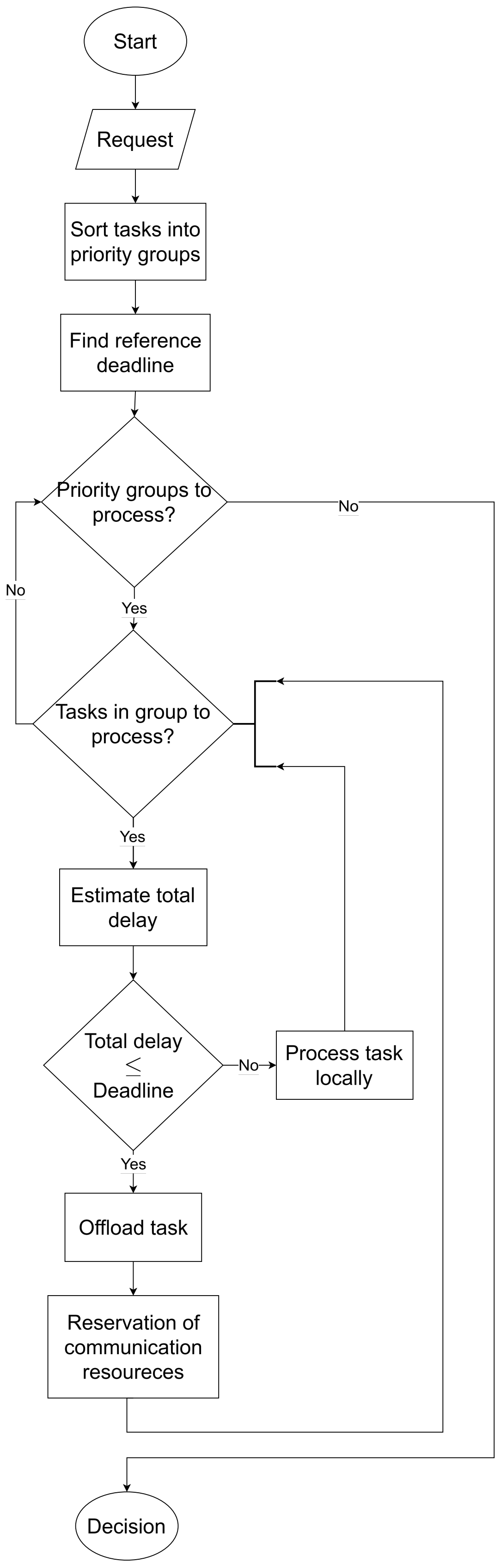


Figure 4.5: Flow chart of the offloading algorithm.

The offloading request sent from the AV is not the only input to the offloading decision algorithm. Uplink and downlink data rates are required to estimate the transmission delay. For this reason we introduce the bitrate monitor. This process collects RAN information from FlexRIC, described in [39]. To exchange information between two parallel processes - the bitrate monitor constantly collecting data and the offloading algorithm in need to asynchronously access them, the Inter-Process Communication (IPC) is used. Through the use of the mutual exclusion (mutex) [40] we ensure that only one process is accessing the shared data at a time. This guarantees integrity of the shared data and prevents runtime errors.

The overall time complexity of the offloading decision algorithm is given by sorting the tasks by priority and is equal to $\mathcal{O}(n \log n)$. It is important to note that n is almost negligible compared to related algorithms, described in this thesis, which process tasks from all connected AVs, aggregating large number of tasks. This proposed algorithm requires little to none computational and storage resources enabling almost indefinite scaling and processing of all the AVs in parallel, making up for the $\mathcal{O}(n \log n)$ time complexity.

Chapter 5

Testing of the Deployed MEC Server

The testing process is structured to ensure a comprehensive evaluation of the communication link between the MEC server, MEC manager and AVs, alongside the verification of the integrated system as a whole. This chapter details both the static and dynamic testing phases, aiming to validate each component's functionality and the overall system's performance under controlled and real-world conditions. The static testing stage involves validating the integration of individual components while the AV is stationary. The dynamic testing validates correct operation of the AV and the MEC system in realistic conditions.

5.1 Static Testing

In this section, we explore the testing of integration with the model AV. This phase is critical in verifying the integrity and inter-functionality of the comprehensive system architecture, involving both hardware components and software systems. We aim to ensure that the interactions between the AV and the server adheres to expected behaviors in a controlled environment, which is pivotal before transitioning to more dynamic and unpredictable real-world conditions.

The initial phase of static testing is conducted on real hardware while the AV remains stationary and involves testing of individual components from Figure 4.4.

1. The internal communication within the Docker Node is tested to verify communication between application containers using Kafka topics.

    - This step is essential to verify that each container can publish and subscribe to the necessary topics without any data loss or corruption.

2. The communication between the Offloading Manager and the Gateway Container is verified.

    - This involves ensuring that the connection is successfully established and messages are correctly identified and forwarded.

3. The distribution of Kafka topic UUIDs from the Offloading Manager to the containers is tested.

    - This is to confirm that unique topics are used by replicated containers, which is crucial for maintaining the integrity and flow of data within the system.

The internal communication within the Docker Node is tested to verify communication between application containers using Kafka topics. This step is essential to verify that each container can publish and subscribe to the necessary topics without any data loss or corruption. Next, the communication between the Offloading Manager and the Gateway Container is verified. This involves ensuring that connection is successfully established and messages are correctly identified and forwarded. Following this, we test the distribution of Kafka topic UUIDs from the Offloading Manager to the containers to confirm that unique topics are used by replicated containers. This is crucial for maintaining the integrity and flow of data within the system.

The communication link between the AV and the Gateway container is another critical aspect that needs thorough testing. In this phase, we ensure that data sent from the AV is correctly received by the Gateway container and that responses are accurately sent back to the AV. This step verifies the reliability and stability of the communication link, which is vital for the overall functioning of the system. Additionally, the communication between RVIZ 2, the ROS Container and the AV is verified to ensure that the visualization tool is able to display data and send checkpoints to the AV. Lastly, the communication link between FlexRIC and the Bitrate Monitor is tested to ensure that bitrate data is accurately collected, transmitted, and transferred to the Offloading Algorithm.

After verifying the individual components, we proceed to test the integrity of the deployed system. The first step in system integration testing is to verify that the offloading data sent from the AV reaches the intended containers within the MEC server. This involves conducting end-to-end testing to ensure data integrity, correct routing, and proper handling by the Gateway container. We also verify that the data processed by the containers is correctly sent back to the AV, ensuring a complete data flow. Following this, we test the offloading requests to ensure that requests sent by the AV are correctly forwarded to and received by the Offloading Manager. The Offloading Manager processes these requests and sends the correct offloading decisions back to the AV. This step ensures that the offloading mechanism works seamlessly, maintaining the integrity and efficiency of the system.

This comprehensive testing process ensures that all components and their interactions are thoroughly validated in a controlled environment. By doing so, we provide a robust foundation for subsequent dynamic testing phases, ensuring that the system performs as expected before deployment in more dynamic and unpredictable real-world conditions.

5.2 Dynamic Testing with the model AV

Dynamic testing represents the phase where the system is evaluated under conditions that closely mimic real-world operations. Before deploying the model AV on a track, we utilize *ROS bags*, which are records of previous routes taken by the AV, to play back the recorded data to reproduce the exact conditions from past runs. These ROS bags are instrumental in verifying the system's functionality without the inherent risks of real-world testing. By replaying these recorded routes, we can observe the system's behavior and identify any serious errors in a controlled environment, ensuring that the AV does not encounter any unforeseen issues that could result in a crash or malfunction.

Once the system behavior is verified through various scenarios using ROS bags, we proceed to on-track testing. This stage involves the manual input of a destination point (goal) in RVIZ 2, which initiates the release of the AV, and verification that the AV reaches its goal. During this phase, we closely monitor the states of the onboard ROS nodes through the Control Node deployed in the DT. This monitoring is crucial to ensure that all nodes are functioning correctly and to quickly identify and address any issues that arise. The interaction between the AV and the MEC server is observed in real-time to confirm that the offloading processes and communication links are working as intended.

Logs are collected from the AV, the Docker Node, and the Offloading Manager throughout the testing process. These logs are vital for post-processing and evaluation, providing detailed insights into the system's performance, any errors encountered, and the efficiency of the offloading processes. By analyzing these logs, we can assess the overall functionality of the system and make necessary adjustments to improve performance and reliability.

Once the system's overall functionality is validated through dynamic testing, we are ready to move on to the experimental stage. This stage involves more extensive and varied testing scenarios to further evaluate the system's capabilities and robustness. The successful completion of dynamic testing ensures that the system is ready for real-world deployment, providing confidence in its ability to handle the complexities and challenges of actual operation.

Chapter 6
Experiments

This chapter presents a series of experiments designed to evaluate the performance and effectiveness of the proposed solution. This thesis proposes a MEC system architecture with an offloading decision algorithm at its core. By deploying the solution on real hardware we can best evaluate the performance of the proposed system. The experiments focus on various critical aspects such as the impact of data rate, offloading decision time, execution time comparison, deadline adherence, and energy consumption. Through these experiments, we aim to provide a comprehensive analysis of the system's capabilities and limitations, thereby demonstrating its potential for real-world applications.

6.1 Video Demonstration

As part of the experimental validation of the proposed MEC architecture, a video demonstration was created to visually present the key aspects and outcomes of the project. This video provides an overview of the system in action, demonstrating the real-time offloading of computational tasks from the AV to the MEC server.

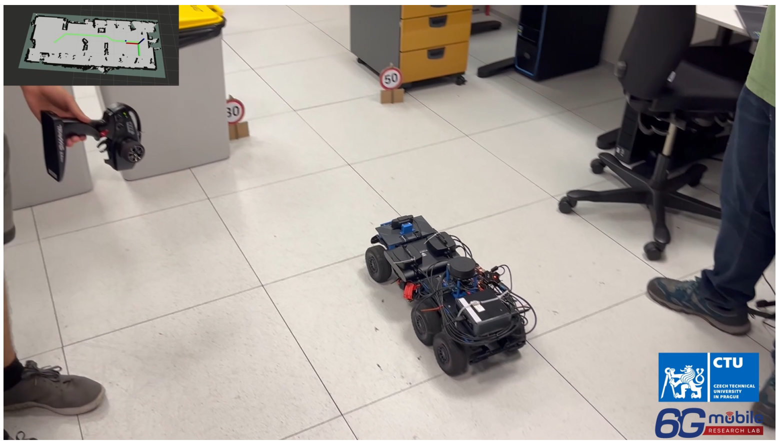


Figure 6.1: Video demonstration accessible from YouTube[47].

The model AV is build and its control algorithms are developed in related thesis. Another part of the AV, the image processing pipeline for road sign detection consisting of two CNNs, is also developed as separate thesis. This work extends mentioned theses with the proposed MEC architecture. Following experiments are performed on the showcased AV.

6.2 Experiment Scenario

Experiments are repeated multiple times to provide enough data for relevant conclusion. We want the conditions to change as little as possible across these experiments to give us unbiased idea of the overall performance. This is impossible to achieve with moving AV as we cannot guarantee that the RRT path planner finds the same path every time, the AV does not steer perfectly every time due to slightest of interference to the motors preventing it from going through same points on the map in each experiments. Therefore, after demonstrating that the AV is able to operate and reach its destination during dynamic testing described in Chapter 5, we use recorded ROS bags for the experiments. The ROS bags provide a way to repeat the experiments as close as possible. The experiments are conducted under different mobile network conditions: with a high quality communication channel (MCS 27) and a low quality communication channel (MCS 5). This is to put into perspective how quality of communication channel affects the offloading process.

We use following hardware and radio setup:

- **AV control unit:** Raspberry Pi 4 Model B with quad-core ARM Cortex-A72 processor, 1.5 GHz, 8 GB RAM, 9.69 GFLOPS [41]

- **MEC server:** Intel NUC11PHKi7CAA2 with i7-1165G7 CPU at 4.7 GHz, 16 GB RAM, 20.4 GFLOPS [42] and NVIDIA RTX 2060 GPU

- **5G mobile network:** Frequency band n78 (3.5 GHz), subcarrier spacing 30 kHz, 106 resource blocks (RBs)

During the experiments, we log the following key metrics:

- **Algorithm execution time:** The time taken by the offloading decision algorithm to process an offloading request and generate a decision.

- **Round-trip time (RTT) of the offloading request:** The total time elapsed from when the AV starts transmitting the offloading request to when it receives the offloading decision. This includes the transmission time from the AV to the MEC server, the algorithm execution time, and the transmission time from the MEC server back to the AV.

- **Task execution time:** The time it takes for the task to process its input data locally on AV put into perspective to the time it takes on the MEC server.

- **Ability of tasks to meet their deadlines:** It is important for hard-deadline tasks to meet their deadlines. We compare local execution, full offloading and the proposal in their ability to provide computational resources for the tasks to meet their deadlines.

- **Energy consumptions:** We monitor the eneregy consumption of the AV's control unit (Raspberry Pi 4) to compare the energy efficiency of local execution and the proposed solution.

To capture these metrics, timestamps are recorded at various stages of the offloading process:

- When the AV begins transmitting the offloading request.

- When the MEC server receives the request.

- When the offloading decision algorithm completes processing the request and generates the offloading decision.

- When the AV receives the offloading decision.

- When the AV starts transmitting the offloading data.

- When the AV receives processed data from offloaded tasks.

6.3 Data Rate Distribution

In this section, we analyze the data rates experienced by the AV during its operation by utilizing the data saved by the Bitrate Monitor. The primary objective of this section is to compare the data rates in channels of different quality that are used for the experiments. The measured data rates are shown in Figure 6.2 in the form of Cumulative Distribution Function (CDF) [43].

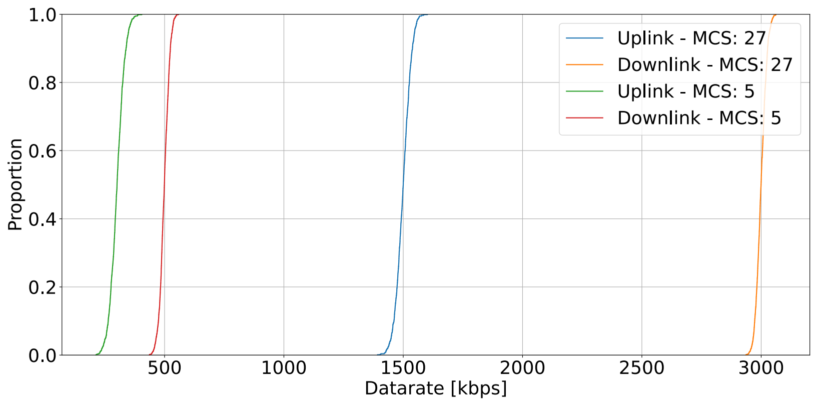


Figure 6.2: CDF of uplink and downlink data rates for different quality channels.

6.4 Offloading Decision Time

In this section, we examine the performance of the offloading decision algorithm by analyzing the delay it introduces. Specifically, we focus on the time taken by the algorithm to calculate the offloading decision and the time the AV has to wait for offloading decision (RTT). These metrics are crucial for understanding the overall latency introduced by the offloading decision mechanism and ensuring that it does not compromise the real-time performance requirements of the AV.

Figure 6.3 shows the CDF of the offloading algorithm's execution time during the experiments. The delay caused by the offloading algorithm processing requested tasks is generally under 100 μs, demonstrating that the computational overhead is low. This is crucial for ensuring that the offloading decision does not introduce significant delays that could affect the AV's performance.

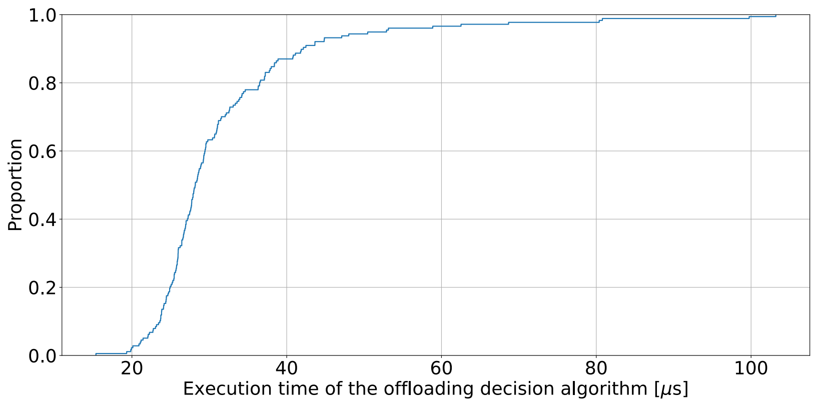


Figure 6.3: Execution time of the offloading decision algorithm.

The Figure 6.4, representing the CDF of total time the AV has to wait for the offloading decision, provides insights into the total latency introduced by the offloading decision process. It includes both the transmission delays and the offloading algorithm's execution time. We see that the majority of the delay is caused by the transmission delay, and the offloading algorithm's execution time is negligible in comparison.

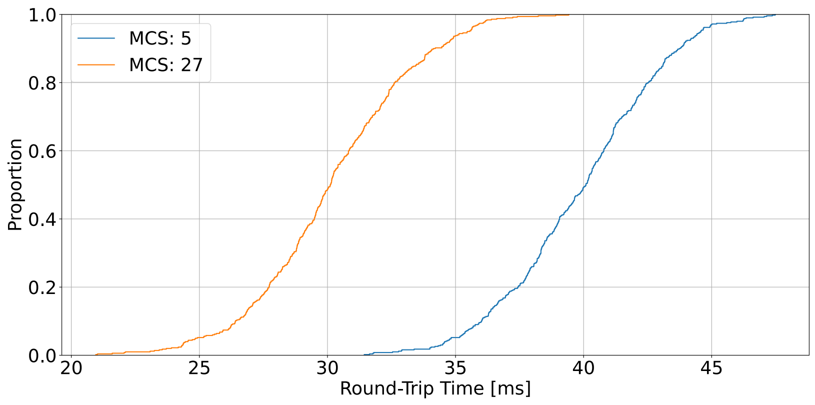


Figure 6.4: Round-trip time from offloading request to offloading decision.

We see that for high quality channel the worst RTT is around 40 ms. As expected, when the channel quality is low the RTT increases. This might cause issues for task with low deadlines as significant part of that deadline might be spent waiting for the offloading decision. The proposed offloading decision algorithm therefore estimates the RTT and considers it in the decision making process.

6.5 Comparison of Execution Times

This section aims to compare the execution times of various tasks when executed locally on the AV versus remotely on the MEC server. By using the same algorithm and parameters for each task, we ensure that the comparisons are relevant and highlight the performance gains achieved through offloading. This comparison demonstrates the enhanced computational capabilities of the MEC server, which allows for the execution of more complex algorithms and longer prediction horizons, thereby improving the overall functionality of the AV.

To conduct this experiment, we selected a set of representative tasks that are offloaded. These tasks include:

- **Model Predictive Control (MPC) [44]:** Used for optimizing the vehicle's control strategies. MPC uses a dynamic model of the AV to predict its future behavior over a finite time horizon. For steering and speed control, MPC optimizes the AV's trajectory by adjusting the steering angle and throttle/brake inputs to minimize a cost function, which includes terms for tracking a desired path and maintaining stability.

- **Pure Pursuit [45]:** A simple yet effective path tracking method used for steering control. The algorithm works by selecting a target point on the desired path a certain look-ahead distance ahead of the AV's current position. The vehicle then adjusts its steering angle to follow a circular path that intersects this target point, effectively "pursuing" the point.

- **Path Planning:** Utilizing algorithms such as RRT and A*. RRT is a path planning algorithm that incrementally builds a tree of possible paths from the starting position to the goal. It explores the space by randomly sampling points and extending the tree towards these points. A* is a graph-based path planning algorithm used for finding the shortest path from a start to a goal position. It combines the benefits of Dijkstra's algorithm [46] and a heuristic approach to efficiently explore the search space.

- **Obstacle Detection:** A computationally intensive task that requires real-time processing.

- **Road Sign Detection & Classification:** An image processing pipeline that locates and classifies road signs can be hugely optimized if executed on a GPU. The pipeline utilizes two CNNs, one for location of the road sign in an image, the other for classification of the located road sign.

Each task is executed locally on the AV and on the MEC server. The execution times are averaged over a range of scenarios to capture the variability in performance. The same set of input parameters was used in both local and remote executions to ensure a fair comparison.

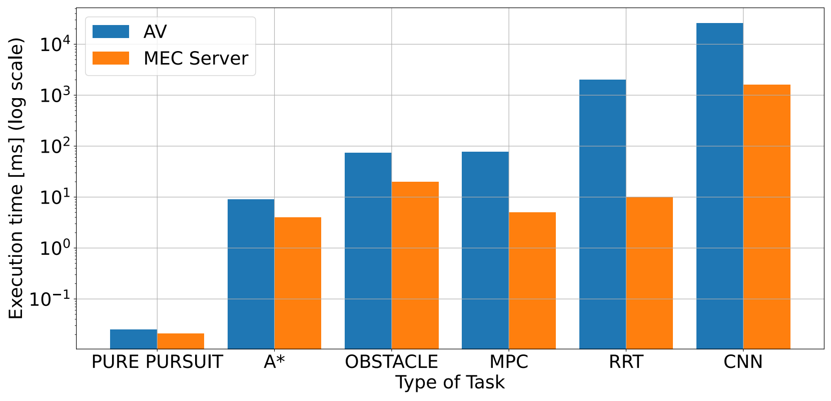


Figure 6.5: Comparison of task execution times (log scale).

The results of the experiment are presented in Figure 6.5. The Figure provides a clear visual representation of the performance differences, highlighting the efficiency gains achieved through offloading. To mitigate the big differences of values, the graph is presented with log scaled Y-axis.

	CNN	RRT	MPC	Obstacle	A*	Pure Pursuit
Relative Gain	16.12	198.7	15.4	3.65	2.25	1.19

Table 6.1: Relative gains due to task offloading.

Table 6.1 shows relative gains by executing task on the MEC server. The most significant difference is for the RRT algorithm used for path planning, which is more complex and computationally intensive than the A* algorithm. The local execution on the AV is limited to using the less accurate A* algorithm due to its computational constraints. The remote execution on the MEC server improves the accuracy and reliability of the path planning process.

Another significant improvement is experienced by the MPC. The enhanced computational power of the MEC server allows for the use of deeper MPC models, providing longer prediction horizons. This results in more accurate and effective control strategies, which significantly improve the performance of the AV.

The relative gain for obstacle detection is not as significant as for some of the other tasks, however, it has to be calculated very frequently. Offloading this task to the MEC server releases computational resource for other tasks.

Although the relative gain from offloading the image processing pipeline is only the second largest, in absolute numbers, it is by far the most time consuming task to execute on AV (over 25 seconds). As the AV is not equipped with GPU, the road sign classification takes too long for the pipeline to be deployed in real environment. However, when executed on the MEC server the execution time drops enough to enable road sign detection in real traffic conditions.

6.6 Tasks Processed within Deadline

The primary objective of this section is to analyze the ability of the tasks to meet their deadline during the AV's operation. This analysis helps to understand the effectiveness of the offloading strategy and the system's ability to meet real-time requirements. We evaluate the ratio of met/missed deadlines for following scenarios: local execution, full offloading (all tasks are offloaded without any decision making logic) and the proposal with high/low quality channel quality.

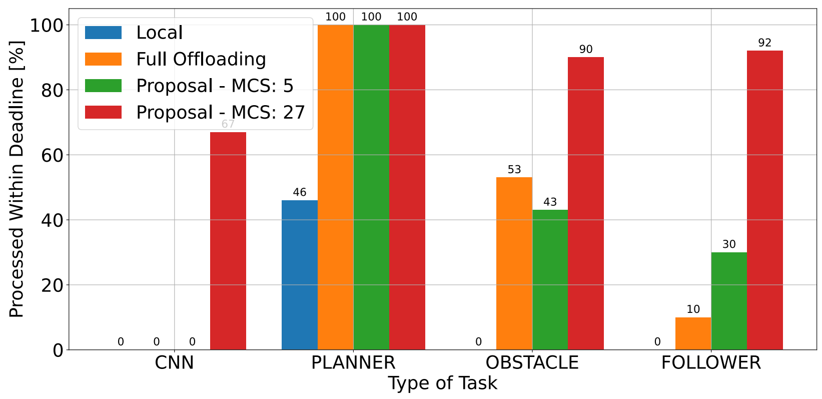


Figure 6.6: Deadlines missed due to insufficient computational resources.

The Figure 6.6 depicts how often the individual tasks are able to meet their deadline. For full local execution we see that the image processing pipeline takes up all of the computational resources so that except for path planner no tasks are able to be processed within their deadline. The path planner is the first task to be executed, and aside from local execution where RRT often misses its deadline, it is always able to meet the deadline as it is the only task executed.

For the full offloading the results are slightly better. However, there is no logic to control if the communication resource available are sufficient to transfer all of the data needed. This leads to long transmission times which cause tasks to miss their deadlines. Tasks like the image processing pipeline require large amount of data to be transmitted blocking the rest of the tasks. This is addressed by the proposed solution. When there are insufficient communication resources (as is the case for MCS 5), only some of the tasks are offloaded. Still a large amount of deadlines is not achieved, because of the low data rates preventing image data to be transmitted along the higher priority tasks. Therefore, the image processing pipeline is always executed locally and only offloaded tasks are able to meet their deadline.

Finally, the best results are achieved by the proposal experiencing high channel quality (MCS 27). We see that the AV is able to process images by offloading them to the MEC server only in this scenario, and consequently, majority of the deadlines is met. This setup yields the best performance of the AV.

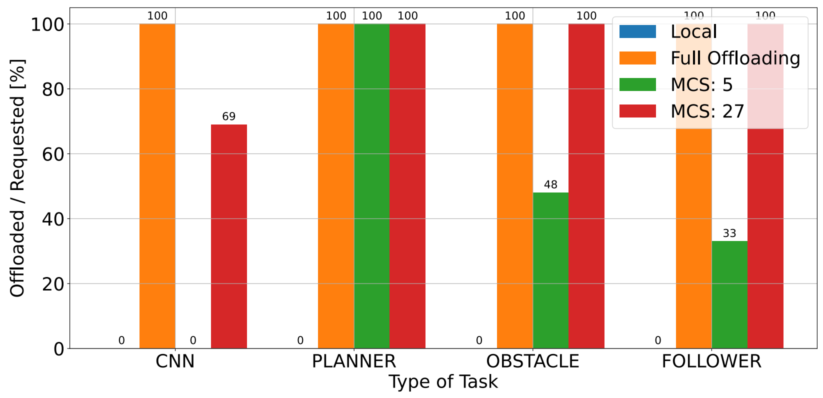


Figure 6.7: Comparison of Offloading Efficiency for Different Task Types.

To help understand why channel quality affects the overall performance of the AV, Figure 6.7 illustrates how the offloading decision algorithm responds to offloading requests under different channel conditions. We see that majority of the missed deadlines is caused by the inability to offload them in time due to low quality of the communication channel resulting in tasks being executed locally. Conversely, while the quality of the communication channel is high it is possible to offload more of the tasks resulting in higher success in achieving deadlines.

6.7 Energy Consumption

This section compares the energy consumption of the AV's control unit when tasks are processed locally versus when task offloading is enabled. By analyzing the energy consumption in different operational modes, we can determine the efficiency gains achieved through offloading and its impact on the AV's overall energy consumption. This analysis is critical for understanding how offloading affects the AV's operational range and battery life.

To conduct this experiment, we measure the energy consumption of the control unit under three different conditions:

- **Idle Energy Consumption:** The baseline energy consumption when the AV is idle and not processing any tasks.

- **Full Local Energy Consumption:** The energy consumption when tasks are processed locally on the AV's control unit, using a ROS bag to ensure consistent conditions.

- **Offloading Energy Consumption:** The energy consumption when tasks are offloaded to the MEC server, using the same ROS bag to maintain consistent conditions.

For each condition, the energy consumption is recorded over the same period, ensuring that the measurements accurately reflect the energy usage in each operational mode.

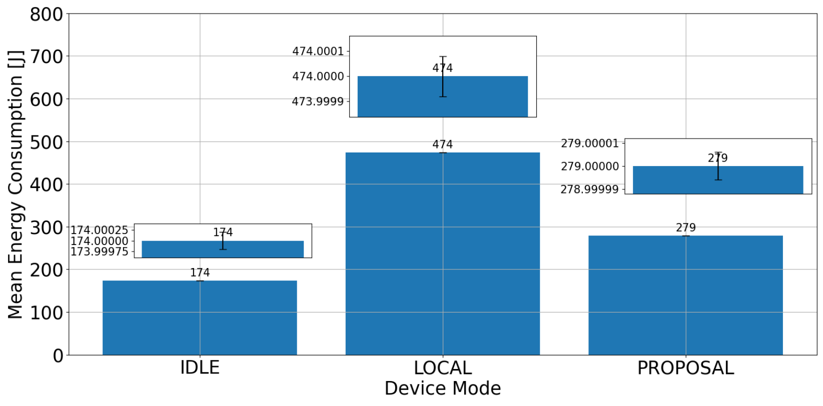


Figure 6.8: Energy consumption in different operational modes.

The results of the experiment are presented in Figure 6.8 comparing the mean energy consumption for three different device modes: Idle, Local, and Proposal. The data reveals significant differences in energy consumption across these modes. Zoomed variances for the three modes are shown above corresponding bars.

The idle energy consumption represents the baseline energy consumption when the device is not performing any computational tasks. When tasks are processed locally on the AV's control unit, the mean energy consumption increases dramatically to 474 J. This is due to the device handling all computational tasks locally, which requires substantial energy resources. By comparing this with the idle energy consumption, we can quantify the energy overhead associated with local task processing. Offloading tasks to the MEC server reduces the computational load on the AV's control unit, the mean energy consumption is reduced to 279 J (58.86 %).

The variance for all of the modes is minimal, indicating that the energy consumption is relatively stable, thus predictable. The depicted variance of proposal is low enough to conclude that the energy consumption is stable even when offloading is used.

Chapter 7

Conclusion

In this thesis, we have developed a flexible and scalable MEC architecture designed to support a large number of AVs with diverse computational needs. The architecture includes a dynamic management system capable of handling fluctuating mobile network demand, ensuring efficient task offloading. The integration of DT provides a stable environment for the offloaded applications. A robust task offloading algorithm is introduced, prioritizing and managing computational tasks effectively under varying conditions of mobile network congestion and resource availability. This algorithm improves the performance of AVs with scarce computational resource by optimizing the use of MEC servers.

The implemented MEC system is thoroughly tested to ensure correct operation in various scenarios. Initially, static tests are performed to validate the integration of individual components. After successful integration and testing of internal and external communication in the system we have continued with dynamic testing. These tests put the AV in operation, validating that it can navigate through tracks with obstacles present.

The proposed solutions is validated through implementation on real hardware. Experiments demonstrated the feasibility of the system in real-world scenarios, showing substantial improvements in computational efficiency, energy consumption, and the ability of tasks to meet real-time processing deadlines. In the conducted experiments, we have measured up to 58 % in energy savings and 60 - 90 % increase in ratio of tasks processed within required deadline.

The system is tested under various mobile network conditions, including low and high quality communication channels. The results indicated that the proposed offloading algorithm effectively managed offloading process even in adverse conditions.

While this research has made improvements in the application of MEC for AVs, there is room for future work. Ensuring the security of data and communications between AVs and MEC servers is critical. Future research should investigate robust security measures to protect against potential vulnerabilities and attacks.

Exploring the use of multi-exit CNNs, which allow early exits at intermediate layers of the network, could provide a balance between computational efficiency and accuracy. This approach could significantly reduce the road sign classification time while keeping the option to leverage the MEC server.

Future work should also explore the possibility of expanding the DT. There is potential to expand the functionality of the DT and much can be gained by introducing interactions between DTs of individual AVs. Information exchange between DTs could bring new features to the AVs improving their performance and efficiency.

Finally, future research should focus on handling handovers, which pose significant challenge for the concept of offloading as the AVs require large amount of data to be transferred between MEC servers.

Appendix A

Bibliography

[1] Sunyaev A. (2020). Cloud Computing. In: *Internet Computing. Springer*, Cham. https://doi.org/10.1007/978-3-030-34957-8_7

[2] Mach P. and Becvar Z. "Mobile Edge Computing: A Survey on Architecture and Computation Offloading," *IEEE Communications Surveys & Tutorials*, vol. 19, no. 3, pp. 1628-1656, 2017, doi: 10.1109/COMST.2017.2682318.

[3] Mao Y., You C., Zhang J., Huang K. and Letaief K. B., "A Survey on Mobile Edge Computing: The Communication Perspective," in *IEEE Communications Surveys & Tutorials*, vol. 19, no. 4, pp. 2322-2358, Fourthquarter 2017, doi: 10.1109/COMST.2017.2745201.

[4] Powell C., Desiniotis C., Dezfouli B. The fog development kit: A platform for the development and management of fog systems. *IEEE Internet Things J.* 2020, 7, 3198–3213

[5] Ahangar M.N., Ahmed, Q.Z., Khan F.A., Hafeez M. A Survey of Autonomous Vehicles: Enabling Communication Technologies and Challenges. *Sensors 2021*, 21, 706. https://doi.org/10.3390/s21030706

[6] Vargas J., Alsweiss S., Toker, O., Razdan R., Santos J. An Overview of Autonomous Vehicles Sensors and Their Vulnerability to Weather Conditions. *Sensors 2021*, 21, 5397. https://doi.org/10.3390/s21165397

[7] Bute M. S., Fan P., Liu G., Abbas F., and Ding Z., "A Collaborative Task Offloading Scheme in Vehicular Edge Computing," in *IEEE 93rd Vehicular Technology Conference (VTC2021-Spring)*, 2021, pp. 1-5, doi: 10.1109/VTC2021-Spring51267.2021.9448975.

[8] Li M., Gao J., Zhao L. and Shen X., "Adaptive Computing Scheduling for Edge-Assisted Autonomous Driving," in *IEEE Transactions on Vehicular Technology*, vol. 70, no. 6, pp. 5318-5331, June 2021, doi: 10.1109/TVT.2021.3062653.

[9] Cui M., Zhong S., Li B., Chen X. and Huang K., "Offloading Autonomous Driving Services via Edge Computing," in *IEEE Internet*

of Things Journal, vol. 7, no. 10, pp. 10535-10547, Oct. 2020, doi: 10.1109/JIOT.2020.3001218.

[10] Ali W.A., Fanti M.P., Roccotelli M., Ranieri L., "A Review of Digital Twin Technology for Electric and Autonomous Vehicles". *Appl. Sci.* 2023, 13, 5871. https://doi.org/10.3390/app13105871

[11] Ndashimye E., Sarkar N. and Ray S., "A network selection method for handover in vehicle-to-infrastructure communications in multi-tier networks," *Wireless Networks.* 26. 10.1007/s11276-018-1817-x.

[12] Verschoor T., Charpentier V., Slamnik-Kriještorac N. and Marquez-Barja J., "The testing framework for Vehicular Edge Computing and Communications on the Smart Highway," 2023 *IEEE 20th Consumer Communications & Networking Conference (CCNC)*, Las Vegas, NV, USA, 2023, pp. 1147-1150, doi: 10.1109/CCNC51644.2023.10060332.

[13] Naudts D., Maglogiannis V., Hadiwardoyo S. et al. "Vehicular Communication Management Framework: A Flexible Hybrid Connectivity Platform for CCAM Services," in *Future Internet*, 2021, 13. 81. 10.3390/fi13030081.

[14] Wu L., Zhang R., Li Q. et al. "A mobile edge computing-based applications execution framework for Internet of Vehicles," in *Front. Comput. Sci. 16*, 165506 (2022). https://doi.org/10.1007/s11704-021-0425-6

[15] Jeremiah S. R., Yang L. T., Park J. H., "Digital twin-assisted resource allocation framework based on edge collaboration for vehicular edge computing," in *Future Generation Computer Systems*, Volume 150, 2024, Pages 243-254, ISSN 0167-739X, https://doi.org/10.1016/j.future.2023.09.001

[16] Peng H. and Shen X., "Deep Reinforcement Learning Based Resource Management for Multi-Access Edge Computing in Vehicular Networks," in *IEEE Transactions on Network Science and Engineering*, vol. 7, no. 4, pp. 2416-2428, 1 Oct.-Dec. 2020, doi: 10.1109/TNSE.2020.2978856

[17] Paranjothi A., Khan M. S., Zeadally S., "A survey on congestion detection and control in connected vehicles," in *Ad Hoc Networks*, Volume 108, 2020, 102277, ISSN 1570-8705, https://doi.org/10.1016/j.adhoc.2020.102277.

[18] Bréhon–Grataloup L., Kacimi R., Beylot A-L., “Mobile edge computing for V2X architectures and applications: A survey”, 2022 *Computer Networks*, Volume 206, 108797, ISSN 1389-1286, https://doi.org/10.1016/j.comnet.2022.108797.

[19] Wu H., Sun Y. and Wolter K., "Energy-Efficient Decision Making for Mobile Cloud Offloading," in *IEEE Transactions on Cloud Computing*, vol. 8, no. 2, pp. 570-584, 1 April-June 2020, doi: 10.1109/TCC.2018.2789446

[20] Zhang J., Wu Y., Min G. and Li K., "Neural Network-Based Game Theory for Scalable Offloading in Vehicular Edge Computing: A Transfer Learning Approach," in *IEEE Transactions on Intelligent Transportation Systems*, doi: 10.1109/TITS.2023.3348074

[21] Cui T., Hu Y., Shen B., Chen Q. Task Offloading Based on Lyapunov Optimization for MEC-Assisted Vehicular Platooning Networks. *Sensors 2019*, 19, 4974. https://doi.org/10.3390/s19224974

[22] “ETSI GS MEC 003 V3.1.1, “Mobile Edge Computing (MEC); Framework and Reference Architecture”,” ETSI, 2022. [Online]. Available: https://www.etsi.org/ deliver/etsi gs/MEC/001 099/003/03.01.01 60/gs MEC003v030101p.pdf

[23] Betz T., Li W., Pan F., Kaljavesi G., Zuepke A., Bastoni A., Caccamo M., Knoll A. and Betz J. (2024). A containerized microservice architecture for a ROS 2 autonomous driving software: an End-to-End latency evaluation. *arXiv (Cornell University).* https://doi.org/10.48550/arxiv.2404.12683

[24] "Intel Smart Edge Open," (2023). [Release-Notes]. Available: https://smart-edge-open.github.io/release-notes/ (accessed May 4, 2024)

[25] "OAI-MEP," (2023). [Source Code]. Available: https://gitlab.eurecom.fr/oai/orchestration/oai-mec/oai-mep (accessed May 4, 2024)

[26] Wang S., Song X., Xu H., Song T., Zhang G., Yang Y., "Joint offloading decision and resource allocation in vehicular edge computing networks", in *Digital Communications and Networks*, 2023, ISSN 2352-8648, https://doi.org/10.1016/j.dcan.2023.03.006.

[27] Zhao L. et al., "A Digital Twin-Assisted Intelligent Partial Offloading Approach for Vehicular Edge Computing," in *IEEE Journal on Selected Areas in Communications*, vol. 41, no. 11, pp. 3386-3400, Nov. 2023, doi: 10.1109/JSAC.2023.3310062.

[28] Đorđević B., Timčenko V., Lazić M. and Davidović N., "Performance comparison of Docker and Podman container-based virtualization," *2022 21st International Symposium INFOTEH-JAHORINA (INFOTEH)*, East Sarajevo, Bosnia and Herzegovina, 2022, pp. 1-6, doi: 10.1109/INFOTEH53737.2022.9751277.

[29] Liu Y., Lan D., Pang Z., Karlsson M. and Gong S., "Performance Evaluation of Containerization in Edge-Cloud Computing Stacks for Industrial Applications: A Client Perspective," in *IEEE Open Journal of the Industrial Electronics Society*, vol. 2, pp. 153-168, 2021, doi: 10.1109/OJIES.2021.3055901.

[30] “ROS Master,” (2018). ros.org, http://wiki.ros.org/Master (accessed May 5, 2024).

[31] LaValle S., "Rapidly-exploring random trees: A new tool for path planning." in *Research Report 9811*, 1998.

[32] Hart P. E., Nilsson N. J., Raphael B., "A Formal Basis for the Heuristic Determination of Minimum Cost Paths". in *IEEE Transactions on Systems Science and Cybernetics*, 4 (2): 100–7, 1968 doi:10.1109/TSSC.1968.300136.

[33] “Docker compose Overview,” (2024). Docker Documentation, https://docs.docker.com/compose/ (accessed May 5, 2024).

[34] “Swarm mode overview,” (2024). Docker Documentation, https://docs.docker.com/engine/swarm/ (accessed May 5, 2024).

[35] “Documentation,” (2023). Apache Kafka, https://kafka.apache.org/documentation/#introduction (accessed May 5, 2024).

[36] Raptis T. P. and Passarella A., "A Survey on Networked Data Streaming With Apache Kafka," in *IEEE Access*, vol. 11, pp. 85333-85350, 2023, doi: 10.1109/ACCESS.2023.3303810.

[37] Hunt P., Konar M., Junqueira F. P., and Reed B, "ZooKeeper: Wait-Free Coordination for Internet-Scale Systems. in *Proceedings of the 2010 USENIX Conference on USENIX Annual Technical Conference*, Boston, MA, USA, 2010, USENIXATC’10, USENIX Association, 11

[38] Maghfiroh H., Santoso H. P., "Navigation of Self-Balancing Robot using Gazebo and RVIZ," in *Journal of Robotics and Control (JRC)*, 2021 Sep;2(5).

[39] Silva M. et al., "O-RAN and RIC Compliant Solutions for Next Generation Networks," *IEEE INFOCOM 2023 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, Hoboken, NJ, USA, 2023, pp. 1-7, doi: 10.1109/INFOCOMWKSHPS57453.2023.10225994.

[40] "Mutual Exclusion Locks", (1999). Marshall D., https://users.cs.cf.ac.uk/Dave.Marshall/C/node31.html#SECTION00311 0000000000000000 (accessed May 19, 2024)

[41] "GFLOPS," (2024). VMW Research Group, https://web.eece.maine.edu/ṽweaver/group/green_machines.html (accessed May 21, 2024)

[42] "Intel Core i7 1165G7" (2024). NANOREVIEW.net https://nanoreview.net/en/cpu/intel-core-i7-1165g7 (accessed May 21, 2024)

[43] Deisenroth M. P., Faisal A. A., Ong C. S., "Mathematics for Machine Learning" in *Cambridge University Press*, 2020, p. 181, ISBN 9781108455145.

[44] Schwenzer, M., Ay, M., Bergs, T. et al. Review on model predictive control: an engineering perspective. Int J Adv Manuf Technol 117, 1327–1349 (2021). https://doi.org/10.1007/s00170-021-07682-3

[45] Samuel M., Mohamed H., Maziah B. M., "A review of some pure-pursuit based path tracking techniques for control of autonomous vehicle." in *International Journal of Computer Applications 135*, 2016, no. 1, pp. 35-38.

[46] Dijkstra E. W., "A note on problems in connection with graphs[J]", in *Numer Math*, no. 1, pp. 269-271, 1959.

[47] "Autonomous Driving with Offloading to MEC - Phase II," (2024). 6Gmobile research lab https://www.youtube.com/watch?v=aPKcAR9Qli4 (accessed May 23, 2024).