MASTER THESIS

Optimal Handover in MEC for an Automotive Application

Eva Karina van den Eijnden

EEMCS/Internet Science and Technology
Design and Analysis of Communication Systems (DACS)

08-12-2020

TNO innovation for life

UNIVERSITY OF TWENTE.

Abstract

With the rapid evolution of automated driving, an ever-increasing number of driving tasks is being taken over by smart vehicles. To operate, these smart vehicles need more information than that provided by their own sensors. Automated vehicles therefore need to communicate with the infrastructure surrounding them, and they need to be able to do it reliably and in real-time to ensure passenger safety.

Traditional cloud computing cannot deliver the required data quickly enough, because the large physical distance between servers and user devices causes a long round-trip time (RTT). Therefore, a new technique must be adopted to fill the existing performance gap; Multi-Access Edge Computing (MEC). In MEC, infrastructure is brought physically closer to the user to avoid having to go through the core network. This reduces the response time experienced by the end user, allowing a myriad of different applications, including automated driving ones.

MEC requires that an automated vehicles' data is handed over from one server to the next whenever the connection quality starts to deteriorate due to physical distance or overloading. In this work, we investigate what the optimal strategy is for handover timing and the connected server. We define the optimal strategy as the strategy that causes the least frequent violations of round-trip time requirements, as this is a vital aspect of safety standards for automotive applications.

We do this using a novel model for MEC implemented in the ns-3 network simulator [21]. Conclusions are based on a replicated set of experiments conducted on an oval track with 100 vehicles travelling 90 to 110 km/h. In our experiments, we consider a single use case for the automotive application; a platooning application that was created at TNO in the context of the European AUTOPILOT project. This provides us with a realistic set of parameters. The experiments test a set of eight different strategies, each comprised of a combination of a metric for connection quality and a trigger. The metrics are the following:

- RTT observed by the vehicle
- Physical distance to the server

There are four different triggers defining *when* to initiate a handover. These triggers are as follows:

- Optimal, handover as soon as a better alternative is found
- Hysteresis, handover when an alternative is found that is at least 15% better
- Threshold, do not handover unless the service level drops below a certain threshold
- Threshold & hysteresis, a combination of the previous two triggers

The results show that the optimal data handover metric for a platooning application is delay (the RTT observed by the vehicle), and that it far outperforms strategies where the metric is the physical distance to the server. Furthermore, the results indicate that the optimal result is achieved by applying hysteresis to the trigger mechanism. Thus, the optimal data handover strategy for a platooning application is the delay-hysteresis strategy.

Preface

In the long process of writing this thesis, I have had to overcome many challenges of varying nature. While not always a happy time, it has been a time of immense learning and personal growth.

I would like to thank the members of my committee for their input and constructive criticism, both of which have improved my work drastically. Thank you Ramon de Souza Schwartz, for your insightful comments throughout this project and even before, during my internship. I appreciate our talks and your assistance. Thanks to Geert Heijenk for helping me see the project from angles I had yet to consider, and to Hans van den Berg, who was able to help me see the forest for the trees when my initial approach to this project was proving unmanageable and I had to switch gears. And thank you, Marten van Sinderen. You joined the project a little later than the others, which has helped remind me to communicate clearly all the little decisions I had forgotten I had made over the course of my work.

During this project I had the opportunity to visit snowy Oulu, Finland and work with the lovely people at VTT. To them, I would like to extend my gratitude for the warm reception and unforgettable experiences, though I have to admit trying *mämmi* was an experience that will stay with me forever... I would particularly like to thank Tiia Ojanperä, Mikko Majanen and Jyrki Huusko for their invaluable insights and input into this project. *Kiitos!*

Finally, a thank you to my wonderful friends and family, who have been my moral support throughout this project, and without whom this thesis might well have remained unfinished.

Contents

Chapter 1

Introduction

Automated driving is evolving rapidly, with smart vehicles taking over more and more tasks that were typically executed by the driver. It is not unusual to have cruise control on a vehicle. Features such as adaptive cruise control and lane-keeping assistants are quickly gaining ground. Multitudes more automotive applications are under development currently, all of them with complex requirements and constraints. These constraints cannot always be met by the current network technology; for example, a maneuver planning application would typically allow 10 ms latency between the moment an object is detected somewhere in the system and the moment the vehicle is updated by the system, based on the desired control update rate [11]. Most modern-day networks and network-based applications depend on cloud computing for complex calculations like these. Although cloud computing can execute the calculations quickly, the delay incurred by traveling the network to and from the cloud is much too large to meet tight delay constraints; according to [10], the four main cloud service providers (CSPs), namely Amazon Elastic Cloud, Microsoft Azure, Google AppEngine, and RackSpace CloudServers, have an average latency of approximately 65 ms measured from 200 vantage points worldwide. The total delay is even higher, as latency is only one among several delay-incurring factors.

This means that to enable automated driving, a new networking paradigm must be adopted. Multi-access Edge Computing (MEC) was designed to create this low-delay network. MEC was first defined by ETSI in 2014 and provides *"the ability to run IT based servers at network edge, applying the concepts of cloud computing"*[17]. The aforementioned servers have a limited computational capacity in comparison to their cloud computing counterparts but have a much larger capacity than user equipment (UE), such as mobile phones, laptops, or vehicles' on-board computational units. This computational capacity can be utilized for a wide range of services. Another defining property of MEC is that it brings computing power closer to the edge of the network, i.e. physically closer to the UE. This can significantly reduce delay, making it an enabling technology for time-constrained applications such as the maneuver planning application.

When a UE is utilizing a MEC service, it is not necessarily stationary. This is an especially vital factor in an automotive use case. Consequently, during the service time, a UE may move from the target area of one MEC server to that of another. The connection between the UE and server will deteriorate or even fail altogether. Unless the UE finds an alternate server to which to connect, the service will be disconnected. In this case, the data associated with the UE must be transferred from the originating MEC to the successor. This process is referred to as "handover". This thesis investigates the optimal approach to this process for an automotive application.

The rest of this document is structured as follows: Chapter 2 will introduce the published work related to this project. Chapter 3 analyzes the problem to be solved, and Chapter 4 describes the design of our approach. Chapter 5 elaborates on the implementation of the experimental environment. Chapter 6 and 7 discuss the experiment results and the conclusions that can be drawn from them, respectively.

Chapter 2

Related Work[1]

Over the last few years, a lot of research has been done on MEC. Part of the effort was focused on defining what MEC is exactly. It is commonly accepted ([1], [17]) that MEC has the following characteristics as compared to classic cloud computing:

- proximity, servers are located close to the end-users
- on-premise, (most) network traffic is restricted to the local network, foregoing the internet's core network.
- low latency, because of the proximity, latency is lower when compared to classical cloud computing
- location awareness, because servers are local, the rough position of end-users is known. This can be used for e.g. geofencing.
- network context information, properties of the network, e.g. radio channel strength, are known, allowing applications to respond to current circumstances.

Naturally, some of the research also focused on the possible applications for MEC; that is, research focused on the problems that MEC can help solve. Furthermore, research was also carried out on a more structural level. These works focus on the underlying techniques for MEC, such as how a UE can best select a server, or how a MEC server should divide up its processing time. The following sections of this chapter will focus on MEC applications and MEC technologies, respectively. The distinction of application or structural research is not always so clear; some papers propose a structural contribution but then also verify their design using a more application-level perspective. However, for ease of reading this divide has been upheld here. The chapter concludes with a section about handover strategies in cellular networks; cellular networks and MEC are closely related, and handover strategies employed in cellular networks can provide a good basis for potential handover strategies in a MEC context.

[1]This chapter was originally written for the preparatory phase of this research, documented in [28], and has been directly copied from that report.

2.1 MEC Applications

This section focuses on the applications that have been designed for MEC, to give the reader an idea of the possibilities. To structure the overview somewhat, we adhere to the classification of Beck et al. [3] They define the following classes of MEC applications:

- Content Scaling
- Local Connectivity
- Offloading
- Augmentation
- Edge Content Delivery
- Aggregation

Each of the classes will be described and exemplified in the following. *Content scaling* applications downscale user-generated content such as images at the edge of the network. Doing this before data traverses the network rather than doing this at the data centre where the file/image will be stored reduces the impact of the application on the core network. This makes an application of this class both easier on the core network and cheaper for the application owner to run. Applications of this type are useful mainly for data-rich applications, such as social networking sites where a lot of images are shared.

Other applications focus on providing *local connectivity*. These applications provide connectivity or a specific service to users in a certain geographical area. The type of connectivity provided can vary; it could be used for geographically targeted advertising, or it could be used for automated vehicles to share their observations of the world. The latter is described in [16]; each vehicle publishes the objects they have detected around them, and a service running in the MEC combines all received data into one Shared World Model (SWM). This SWM is then communicated to each automated vehicle, allowing them to "see" beyond the reach of their own sensors. The paper describes another local connectivity use case; Platoon Management. In this use case, automated vehicles follow one another at a close distance automatically. This reduces the chance of traffic jams; vehicles are closer together, leaving more space for other road users. Vehicles in a platoon also accelerate and brake simultaneously, assuaging the harmonica effect wherein each vehicle must brake harder than its predecessor, eventually causing traffic to come to a halt. However, these platoons need to be managed as a whole. A MEC server provides a solution here; it can provide the overview and can accommodate the highly localized (and time-constrained) nature of the application, whereas cloud computing would put undue strain on the core network as well as the application.

Offloading is one of the most commonly researched classes of MEC applications. The premise is that UEs have limited computation power and battery life, but MEC servers have both in relative abundance. Therefore, the UE can offload computations to the MEC server, conserving energy and bypassing the limit on computation power. Because the MEC server is physically close to the UE, the incurred delay is in an acceptable range. This idea has been received enthusiastically by the research community; many papers have been published describing various scenarios of how best to capitalize on this new development. A few examples are [20], where the focus is on lowering the delay in a mobile gaming scenario, and [2], which provides a strategy for making the decision of whether or not to offload a certain computation.

Augmentation applications provide extra information (e.g. the number of connected UEs or the available bandwidth) to application service providers (ASPs) so that ASPs can adapt their service strategies in real time, such as Tran et al [27]. Although in their work they refer to context-aware services rather than augmentation, the two concepts are essentially interchangeable. The paper proposes a resource management platform which takes into consideration the augmented data and applies this framework to a set of three use cases to demonstrate its effectiveness.

Applications in the *edge content delivery* class provide cached content delivery from the edge of the network. The aim is to drastically decrease the latency experienced by the user for the most popular applications. The applications that benefit most from this approach are typically media streaming applications. In their work [14], Malandrino et al use a large data set to determine the best caching architecture, i.e. the best place to cache data, in a MEC-enabled environment. They consider four cases: caching in base stations, in base station rings, in aggregation-layer pods or core-layer switches. They conclude that in cases with localized content, such as navigation data, MEC provides a good solution, but when the content is less localized, a more centralized approach works better.

The final class of applications use MEC for *aggregation*. These are applications that aggregate related data from devices in the same geographical area (e.g. V2V or wireless sensor networks communications) before providing the aggregated data to a (cloud-based) server. Xiao et al [31] describe an architecture that aims to allow large-scale crowdsensing by making use of MEC. The use case they use is that of a young child gone missing in a crowd; using the cameras of (consenting) bystanders, the child could be localized before they even realize their parents are gone. However, an application like this requires an enormous amount of processing power; if the images of each phone must be processed in the cloud, this would put an enormous strain on the network, making the widespread use of this application impossible. With MEC however, sources can be analyzed locally, making the processing more distributed and keeping the flow of data off the core network. Theoretically, this aggregation of data enables the scaling up of crowdsensing applications.

2.2 MEC Technologies

Another branch of research focuses on the underlying technologies in MEC. Reading through the papers in this area of study reveals four categories in which the research is focused. Each is listed and detailed in the following.

- Resource allocation
- Cooperative computing
- Server selection
- Handover technique

Resource allocation research focuses on finding the best strategy for the allocation of computational capacity and/or radio capacity to UEs based on a certain service aspect, e.g. energy consumption or delay tolerance. [22] refers to this as resource management and proposes a time-division multiple access (TDMA) based approach as part of their work on MEC on fiber-wireless (FiWi) networks, networks whose topology partially consists of wired links and partially relies on wireless links. They conclude that 'obtained results show the significant benefits of MEC over FiWi networks'. Satria et al [23] propose two distinct recovery schemes for overloaded MEC servers. Each scheme provides a way for traffic or jobs destined for the overloaded MEC server to be redirected, either to neighbouring MEC servers or by using nearby UEs as relay nodes to reach neighbouring MEC servers. By redirecting incoming jobs like this, the overloaded server gets the opportunity to recover and resume service.

Cooperative computing refers to UEs not only making use of the compute power of MEC servers, but also of one another's. This is often referred to as fog computing in literature, although this term is not well-defined; it may refer to only offloading to a MEC server, to offloading to both MEC servers and other UEs, or only to offloading to other UEs. Cooperative computing is described in [7], in which a system is developed for vehicular fog computing. The paper investigates several scenarios in which moving and/or parked vehicles can be used for cooperative computing. Tran et al [27] propose a collaborative MEC system that utilizes both MEC servers and UEs and test the system in three different use cases.

Server selection research focuses on finding the best MEC server to connect to from a UEs perspective. Some research connects to the server that is physically nearest ([6], [18]), others focus on cost and leave the definition of cost an open problem ([29], [30]).[26] migrates to a new MEC server only if the total amount of core network traffic is less with migration than it would have been without. The traffic generated by the migration itself is included in this calculation.

Handover technique is closely related to server selection. When a UE determines the current MEC server is no longer the optimal one, a handover must take place. Handover research focuses on how best to execute the migration from one MEC server to another when needed. MEC applications typically work using either virtual machines (VMs) or containers (such as Docker) in the relevant MEC server, and a handover action means migrating the relevant VM/container to the new server. [13] compares the handover performance of VMs to that of containers and concludes that container-based handover experiences less service downtime with each of the four tested applications (game server, high random-access memory (RAM) application, video streaming, and face detection). Even in the least favourable comparison, use of containers leads to more than four and a half times less service downtime as compared to a VM-based approach.

To answer our research question - What is the optimal handover strategy in an automotive MEC use case? - we will need to define exactly what we consider to be a handover strategy. In this work, we consider a handover strategy a combination of two elements: the selection of the optimal MEC server, and the decision of when to make the switch. Our research will therefore mainly take place in the domain of server selection, but will also partially consider the handover technique sub-domain. There are several papers in this area of research that are related to our research question, though none explicitly consider automotive use cases. We provide a concise summary of each.

Heinonen et al [6] claim that to meet 5G latency requirements, applications and core network functions must be optimized, as well as decision-making algorithms and mobility management. They attempt to do this by using a network slice that instantiates virtual network functions (VNFs) at the cloud edge. As these VNFs run in a virtual machine, it is possible to place them in optimal locations, even if those locations change dynamically. In terms of mobility management, the work selects the optimal MEC server during the UE attach procedure; this can either be the physically closest one or can take the network state into account. The latter approach is described in a separate paper by the same authors [9]. In this work there is a separation between radio handover and handover of the MEC server (from here on referred to as data handover); this ensures that the work is (also) applicable for scenarios in which the MEC servers are not co-located with base stations, or situations in which not every base station has its own MEC server. The optimality of the current MEC server is re-evaluated during each handover procedure; however, an optimal strategy is not determined, rather a number of suggestions are made. In the work proposed in this document, determining what makes a server 'optimal' will be a significant aspect of the research. Heinonen et al also provide a performance evaluation of the current situation; they conclude that current handover procedures are not sufficient for low latency services, especially in the context of 5G. They suggest this as an area of further study. This work does not attempt to resolve this, but focuses on the performance of a simple, unoptimized, handover procedure.

Machen et al [13] consider the live migration of stateful applications. They present a layered migration framework using incremental file synchronization. The framework splits the architecture into three layers: the base layer (containing the guest OS and kernel, but no applications), the application layer (containing an idle version of the application and application-specific data) and the instance layer (containing the running state of the application). A copy of the base layer is stored on every MEC server so that this does not have to be transferred in a handover action. The application layer is instantiated in every MEC that is currently running that application; it may be necessary to transfer this layer of an application to a MEC where the application is not yet running. Finally, the instance layer must always be transferred in the case of a handover. During a migration, the application layer is transferred if necessary while the application is still running; then the service is suspended and the instance data is transferred before the application is restarted in the new server. This significantly reduces the amount of data to be transferred while the application is suspended. Due to space limitations (it is a poster work), no mobility model or other experimentation details are provided. This work focuses on *how* to transfer application data, whereas our proposed work will focus on *when* and *where*. However, using the approach mentioned in this work could improve performance, making certain handover tactics more or less suitable for certain applications.

Farris et al [5] propose an approach to better support user mobility for container-based stateless micro-services. They consider a system architecture in which there is a Mobile Edge Orchestrator (MEO), as well as MEC servers. The MEO has an up-to-date view of network state, MEC server status, and user workload. It decides which applications to deploy in which MEC server, as well as when to relocate a particular application to another server. It is assumed that the relevant context data is available in individual MEC servers, that Docker containers are used, and that there are data volumes (DVs) available so that data remains intact even if the application/Docker instance is destroyed. The handover procedure works as follows:

- DV is synchronized between the source and target servers (this is done periodically, not only during handover)
- Service is stopped in the source server

- A final DV sync is executed
- Service is restarted in the target server
- User traffic is switched from the source server to the target server
- Container in the source server is destroyed

This provides a reliable handover; if the procedure fails, the system can roll back and resume service on the source server before retrying the migration. However, the periodic synchronization also incurs a cost per service in terms of a larger number of containers to deploy, duplicated storage needs and back-haul link congestion; it is therefore important to manage the number of secondary instances wisely. Farris et al did a performance evaluation using a small-scale test bed with two workstations as MEC servers and a UE. They compare the described proactive handover results to those of a reactive approach and find that a proactive approach results in a smaller total migration time that stays stable as the volume size increases, while the migration time for a reactive approach increases more or less linearly as the volume size increases.

Plachy and Becvar [18] propose a handover approach with mobility prediction that uses distance as the sole metric for VM migration decisions. It considers an offloading application as the use case. The solution consists of two algorithms. The dynamic VM placement algorithm checks whether there is a better VM to process a job before starting on that job. The second algorithm, named PSwH enhanced with mobility prediction, is used to select a suitable communication path. A handover between MEC servers is executed if it is profitable to the UE from an offloading perspective. The proposed algorithms cooperate by placing the VM before the UE starts offloading; this curbs the handover delay incurred by migrating during an offloading operation. The dynamic VM placement algorithm is therefore started by the UE in between two offloading actions if and only if the signal-to-noise ratio is below a certain threshold. The set of possible candidates consists only of servers which are not overloaded and to which there is an adequate connection. The algorithms' performance was evaluated using MATLAB; the proposed approach was compared to the authors' previous work as well as two other approaches, and it was found to be the most optimal in terms of offloading delay. In terms of UE energy consumption, however, it is outperformed by some of the competitors, which suggests that for some applications the proposed algorithms might not provide an optimal handover tactic.

2.3 Cellular Handover

Although MEC is a relatively new concept, not all aspects of its technological makeup are new. Concepts such as handover have been applied in cellular networks for many years, and an abundance of research has come with it. In cellular networking, handover during active service time is less frequent than it is expected to be in MEC; while in automotive MEC a UE has a continuously active connection to a server, in cellular networks such a connection only exists when making a call. However, it can happen that a UE moves from one base station to another during a call. We describe here the work done in this area of cellular networking.

In cellular networks, there are a few distinctions between types of handovers. They are discussed in [24] and outlined in the following.

Horizontal vs Vertical handovers can either take place between two structures of the same network (horizontal), or between two different networks (vertical). An example of vertical handover could be a UE transferring from an 802.11p to a 5G network. In our work, we will consider a homogeneous network, so there will be only horizontal handovers.

Intra-cell vs inter-cell refers to the physical areas a network is made up of, called cells. Each cell is serviced by at least one base station. Intra-cell handover means the horizontal handover takes place within the cell, while inter-cell means the signal is handed over to another cell. Intra-cell handover is used to diminish inter-channel interference when moving around within a cell. Inter-cell handovers are initiated when a UE is starting to move out of a cell to another one.

Hard vs soft refers to the strategy used when handing over. In a soft handover, a connection to the new cell is made before the connection to the old cell is relinquished. In a hard handover, this is not the case; the old connection is severed before the new one is made. Because a soft handover puts more strain on the resources, many networking solutions use hard handovers instead. The proposed work will do the same.

Though there are different types of handover, each of these takes place in the same fashion. Four phases can be distinguished according to [24]:

- Measurement - this stage consists of doing measurements. In a cellular context, the signal strength is measured at this point, but in MEC another metric could be chosen.
- Initiation - the decision of whether or not to hand over is made in this phase.
- Decision - if there is a need for handover, a decision is made to which channel to hand over. This decision can be made by the UE, the surrounding base stations or by the network and the UE together.
- Execution - the phase in which the actual handover process takes place.

Works on cellular handover also discuss a number of performance metrics. The metrics below are mentioned in both [24] and [19].

- Call blocking probability - the probability that a user attempting to make a new call is blocked.
- Handover blocking probability - the probability that a handover procedure fails.
- Handover probability - the probability that an active call will require a handover before it terminates.
- Call dropping probability - the probability that a call is dropped due to a failed handover.
- Probability of unnecessary handover - the probability of initiating a handover when the channel quality is still adequate.
- Rate of handover - the number of handovers performed by a base station per time unit.
- Interruption duration - the amount of time during a handover that the UE is not in connection with either base station.
- Delay - time between the initiation of a handover and its completion

These metrics are clearly meant to be used in a cellular context, and some of them might not be directly applicable in a MEC context. However, these metrics can still form the basis of a set of metrics for MEC. For example, although a MEC system will not have call blocking, a server that is overly busy could refuse service to a new UE, so a MEC metric could be 'service blocking probability'. Similarly, a server might drop a UE that is already in service if it becomes clear that the server will be unable to meet the UEs delay constraints. The remaining metrics in the aforementioned list can be used in the same way in a MEC context as in a cellular one.

In a cellular system, a handover will take place if the channel quality becomes insufficient. In a MEC environment, it is possible that other metrics would be more appropriate; this will be part of the research. However, a deliberation from cellular handover that also applied to MEC is the following: *when* should handover take place? [19] discusses the following five options:

- Optimal
- Threshold
- Hysteresis
- Hysteresis and threshold
- Scheduling

The first option is to always connect to the *optimal* base station. This will always give the optimal connection, but if the values for the old and new base station are close together and subject to some speculation, which base station is optimal might fluctuate at a considerable rate. This translates into route flapping, even if the old connection is still adequate. To prevent this, there are two possible approaches: using a *threshold* and only handing over when the threshold is exceeded, or by applying *hysteresis*. When using the latter, the UE only hands over when another MEC is stronger by a certain margin. To compound the effect, both of these measures can be combined so the UE only hands over when the connection is no longer good enough *and* a viable candidate has been detected. Finally, it is possible to *schedule* handovers: by predicting when and where a UE will be handing over, the network can plan for this event before the service starts to deteriorate.

Chapter 3

Problem Analysis

Where the previous chapter describes the existing research on MEC, this chapter explains what we will add to that existing body of research, and the motivations for this choice.

The research proposed in this report will be executed in the context of automotive applications. Automotive use cases are especially interesting from a handover perspective for three reasons:

- Most research on handover strategies does not consider UEs that move at **high speeds**. At higher speed, the 'grace period' in which a UE is on the precipice of handover but does not require direct action yet is much shorter.
- Vehicles' mobility is predictable, as they are confined to the road and tend to drive at the speed limit or the maximum achievable speed. This makes for relatively simple **mobility prediction**.
- Some vehicular applications are **safety critical**. E.g. an obstacle-detection application has a very low delay tolerance; this calls for a high-performance handover mechanism to avoid accidents.

The previous chapter demonstrates that a lot of work has already been done concerning MEC. However, although there are works proposing *how* to handover, there are no works discussing *when* a UE should handover from one server to the next. Cellular networking provides some best practices here, however it cannot be assumed that cellular networking concepts are also suitable for MEC. There are a number of significant operational differences between the two domains, as listed below:

- **Number of handovers**. In many MEC scenarios, the UE is continuously exchanging messages with the server. This is different from a cellular scenario, where this continuous exchange is only in place when a call is in progress. When there is no call in progress, no handover need take place. Therefore, a MEC application will handover every time a new eNB is reached, but this is not the case in a cellular system.
- **Consequences of failed handover**. In a cellular system, a failed handover can cause an ongoing call to be dropped. In a MEC system however, a failed handover has more severe consequences; it can lead to a safety-critical message being dropped or delayed. In a worst-case scenario, this could cause a car to crash.
- **Elasticity**. A MEC application is more adaptive to changing channel quality. If the connection deteriorates for a moment, a MEC application can recover. On the other hand, if the connection momentarily deteriorates in a phone call, this will result in a call being dropped entirely.

These factors are significant enough that we have dedicated this work to finding out what the most successful handover strategy would be in an automotive MEC scenario. In order to do so, we must first determine what make a strategy successful or unsuccessful, and how to do an evaluation of various strategies. Furthermore, as we suspect that the best data handover strategy may vary between applications, we ask ourselves by which properties we can group applications in order to predict the type of handover strategy they require. With this, we have structured the research along the following research questions:

What is the optimal handover strategy in an automotive MEC use case?

- What is an appropriate definition of 'optimal' in this context?
- What is the best way to evaluate different handover strategies?
- What classes of MEC applications can be distinguished in light of handover strategies?

Chapter 4

Research design

This chapter covers the design that we made to execute the research. We will first discuss the chosen definition of an optimal handover strategy, followed by a discussion of the handover strategies that will be tested. Finally, it will cover the method of evaluation, including the use case to be employed for the experiments, and the classes of applications that can be distinguished by handover strategies.

4.1 Definition of optimal

It is crucial to have a good definition of what an optimal handover strategy entails. In automotive applications, delay tends to be a vital performance measure. Many other performance measures, such as channel throughput and server queuing time, are reflected in the total delay an application perceives between issuing a request and receiving a reply. This makes the round trip time as measured by the UE a good indication of overall system performance.

Furthermore, most applications have well-defined requirements for delay tolerance. If these requirements are not met, application performance will degrade. For example, a media streaming application might falter, or an autopilot application might be unable to command the vehicle to brake in time for a newly detected obstruction.

It is insufficient to merely consider the average RTT for our definition of optimal. In a MEC scenario where traffic is bursty, the average RTT might be quite low, while there are spikes at the times where a lot of requests are being sent at once. These spikes can cause one or more messages to violate the RTT requirement, leading to performance degradation. This is an especially serious issue in safety-critical systems.

We will therefore consider the optimal handover strategy that has the lowest probability of RTT requirement violation. The average RTT is considered a secondary determinant, i.e. if two strategies cause the same number of violations, the strategy that provides the lowest average RTT is the optimal one.

4.2 Evaluation of strategies

To determine the optimal handover strategy, a process for the evaluation of these strategies must be established. A real-world test bed would be the most accurate way to do this, but is not an achievable method in a project of this size. Instead, we turn to the alternative: simulation. There are several network simulators freely available. Of these, ns-3 [15] has been selected. The wide array of readily available mechanisms and protocols, combined with the large community and resource availability were decisive factors in making the choice.

Although ns-3 has some simple mobility models such as random walk and constant velocity, it does not have a built-in way to model more complex vehicle mobility. However, it can couple with the Simulation of Urban MObility (SUMO) tool [12]. We used this tool to generate mobility traces that are then coupled with ns-3 in the offline mode.

For statistical accuracy, each test of a handover strategy is repeated ten times. For each run, all different strategies are run under the same mobility trace and with the same seed for ns-3's random number generator. This ensures that within a single run, mobility and random number generation cannot be the cause for differences in the strategies' results. Both the trace file and the seed are changed between runs to assuage effects of certain mobility patterns and/or random number orders.

In our experiments, there is a distinction to be made between radio handover and data handover. Radio handover is the handover of radio resources from one eNB to another as a UE moves along the track; it is handled automatically by ns-3's LTE module. Data handover, on the other hand, is the handover from one MEC server to another based on the criteria set by the handover strategy. This means that a UE is not confined to using the MEC server that is associated with its current eNB, but can handover to another server separately from radio handovers. When we speak about a handover strategy, we therefore speak of a strategy for data handover.

Scenario

The experiments are set in a highway scenario. This choice was made to reduce the effect that factors such as traffic flow and mobility prediction have on the outcome of the experiments. A simple traffic flow should reduce these kinds of effects, although in the future it might be interesting to see how handover strategies function in a more complicated (e.g. urban) setting.

The track is a simple two-lane oval shape, with the network infrastructure located in the area enclosed by the road. The aforementioned network infrastructure consists of three eNBs, which are evenly spaced along the road so the entire area has adequate LTE coverage. A MEC server is co-located with each eNB. It is expected that in real-world situations, a MEC server will be connected to multiple eNBs, but to restrict the scale of the experiment and yet make handovers from one server to another possible, we have chosen this approach. A visual representation of the physical layout can be found in Figure 4.1.

Each MEC server has a different service capacity, i.e. a different maximum service rate in jobs/second. This creates a heterogeneity in waiting times that ensures that in experiments that focus on optimizing RTT the UEs do not always connect to the MEC server that is physically closest. If the service capacities were homogeneous, each MEC server would be connected to an equal portion of the UEs, leading to nearly identical waiting times at each server. With the delay experienced in each server being equal, the main factor increasing the RTT for a UE would be whether or not a message can use the speed boost provided by using MEC; the effects of different handover strategies would not be clearly visible. Furthermore, it is not realistic to assume that in a real-world scenario all MEC servers would have the same computational capacity; it is likely that hardware would differ between vendors.

Our scenario considers 100 vehicles; although this does not utilize the full road capacity, it is thought to represent a moderately busy road that generates enough network traffic to provide reasonable results, while not straining the simulator too much and dramatically increasing experiment run time.

Use case

In this work, we test a single use case; a platooning application for automated vehicles. Such an application has already been developed by TNO, ensuring that we can use realistic variables for the different experiment parameters.

A platooning application enables vehicles to travel in a platoon; this means that the vehicles intercommunicate and maneuver as a single entity. Doing this allows the vehicles to drive much closer together without compromising safety, which improves traffic flow and can help to reduce traffic jams.

TNO's platooning application uses both vehicle-to-vehicle and vehicle-to-infrastructure communication. In our experiments, we will only consider the vehicle-to-infrastructure communication. This type of communication is used to create a collective perception; a common understanding of vehicles and other objects in the vicinity. Each Collaborative Perception Message (CPM) contains a list of object IDs and their locations. This means that the size of a single message can vary strongly depending on how busy traffic is at a given time. However, in our experiments the size of a message has no consequence; it does not cause delays in sending or job processing. In our experiments we therefore assume each message received by a UE is 256 bytes long. CPMs are sent to the vehicles at a frequency of 10 Hz.

The platooning application has a low delay tolerance, making it an interesting test subject in terms of performance. For our experiments, we have set the requirement that RTT measured by the vehicle must not exceed 30 ms. This is less strict than the maneuver-planning application requires in [11], however vehicle-to-infrastructure communication in the case of platooning is not safety-critical and therefore some leeway can be allowed. Furthermore, 5G-MOBIX preliminary results show that the average RTT *without* the use of MEC is approximately 30 ms; setting this as the maximum acceptable value means that MEC must outperform traditional cloud computing.

The application is stateless; when transferring from one server to the next, there is no UE data that needs to be exchanged between the servers as they only require a list of current clients. It could be interesting to test multiple types of applications, but due to time constraints we leave this to future work. Section 7.2 elaborates more on this.

In summary, the platooning application that is our use case has the following characteristics:

- Message frequency between MEC server and UEs - 10 Hz
- Message size - 256 bytes
- Delay limit - 30 ms
- Stateless

Model

To be able to run our experiments, we made a model of the scenario. The model of the track is quite simple: the long edge of the oval is 12 kilometers long, and each of the turns is comprised of a semicircle with a 50-meter radius. This enables the vehicles to take the turns at a speed of 100 km/h, so they can maintain their driving speeds. Ensuring vehicles do not have to brake for the turns avoids traffic buildup and ensures the vehicles remain evenly spread over the track. The total length of the track is approximately 24,3 kilometers and has two lanes that are both traversed clockwise.

The infrastructure model is derived from the 5G-MOBIX test site in Helmond, where there are a single eNB and MEC server covering 4 km of road. The virtual test site has been extended so that handovers can and will take place. Infrastructure behavior is modeled by the standard LTE library of ns-3, our chosen network simulator. This library has implemented the standardized protocols and specifications of real-world LTE modules.

Radio handover is handled by ns-3 using a very sensitive trigger. Data handover is handled by the applications written for these experiments; more detail about this will follow in Chapter 5. There is a MEC server co-located to each eNB. That means that each eNB has a MEC server connected to it through a link that is very fast and lossless. In our experiments, we assume that this link introduces no delay. Furthermore, we assume that the application we are modelling is already running on each server, so that only the user data needs to be transferred. This ensures that there does not need to be a complicated data handover mechanism involving several stages, which will make the experiment results more straightforward to interpret.

If a UE is connected to the MEC server that is associated with its current eNB, mobile edge computing can be engaged. The link delay depends on the network configuration and can range up to 10s of milliseconds if the MEC server and eNB are physically far apart. In our experiments we assume that the eNB and the MEC server are co-located and the link incurs no link delay.

However, if a UE connects to a MEC server that is *not* associated with the UE's current eNB, mobile edge computing cannot be used. Instead, the message will have to go through the regular core network. Measurements from preliminary experiments done by 5G-MOBIX at their Helmond test site indicate that this approach is on average 15 ms slower than its MEC counterpart. Therefore, in our experiments, this is the value of the network delay incurred by a non-MEC message exchange.

The vehicles' mobility is modeled in SUMO. Each vehicle uses Krauss' car-following model

[8]. In short, this ensures that each vehicle will attempt to drive the pre-set maximum speed while maintaining enough distance that a crash will not ensue should its predecessor brake. It depends on three parameters: the vehicles' maximum speed, the vehicles' maximum acceleration and the vehicles' maximum deceleration. In our experiments all vehicles are configured identically. The speed limit is set to 100 km/h or 27.8 m/s to mimic the driving conditions at the 5G-MOBIX test site. Vehicles will drive up to 10% slower or faster than that, making the range of effective velocity 90 to 110 km/h. The acceleration and deceleration parameters are set to 3.5 m/s^2 and 2.2 m/s^2 respectively. This was done at the recommendation of SUMO literature [25].

The vehicles use SUMO's default lane-changing model. It is quite complex, but in essence allows vehicles to change lanes if and only is there is enough space between it and its predecessor and follower in both the current and target lane to do so safely. For a full description refer to [4].

At the start of the experiment, vehicles start driving at the top left corner of the track. They start out driving 0 m/s and accelerate at their maximum acceleration until they are going 27.8 m/s. Vehicles are released from the starting point 9 seconds apart. As it takes $24300/27,8 \approx 900$ seconds for a vehicle to complete a full lap of the track, this means that the last vehicle departs just as the first vehicle completes its first lap. This aids in achieving a steady-state for the mobility aspect of the experiments, where the vehicles are evenly spread out over the track. Experiment measurements begin 1000 simulated seconds after the first vehicle leaves, when the steady-state has been achieved.

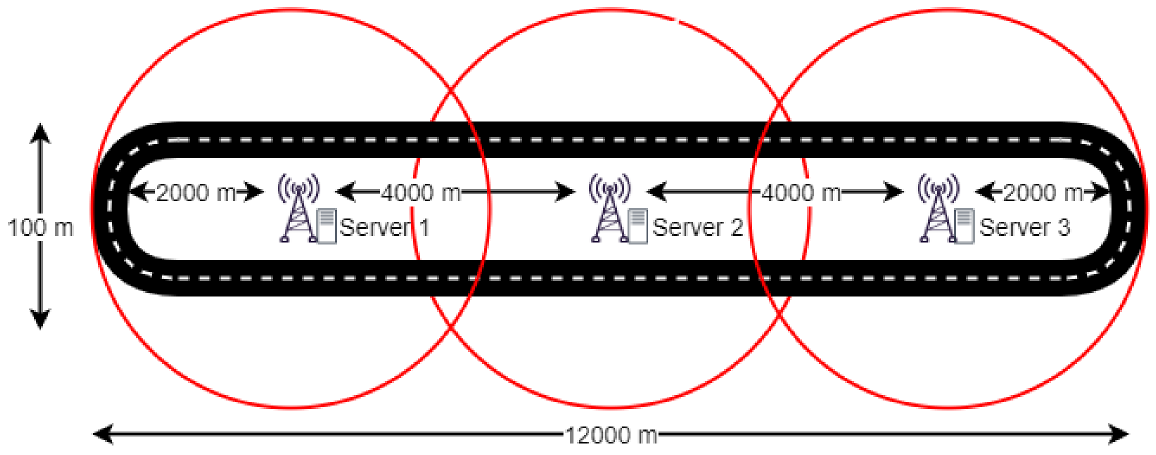

Figure 4.1: *Visual representation of the virtual test site. Not to scale.*

4.3 Handover strategies

We have seen in Section 2.3 that cellular systems can use five types of events to trigger a handover. These triggers are all based on some metric, a concept of what a better connection is. A data handover strategy therefore consists of two elements; the metric and the trigger.

In radio handover, the metric is usually a measure of the quality of the connection, e.g. the Received Signal Strength Indicator (RSSI). In data handover we therefore need not consider this. What *is* an important measurement of the connection quality depends on our definition of an optimal strategy. As we have previously determined that the **total delay, or RTT**, is the most relevant factor, we will use this metric as a component of our handover strategies. We will also run experiments using another metric: **distance**. Connecting to the nearest MEC server should normally result in the smallest amount of link delay, thereby reducing the RTT. However, this metric fails to take into account waiting times at the the server. That is, if the closest MEC server is overloaded, a delay-based strategy will cause a UE to avoid this server, but a distance-based strategy will not. However, a distance-based strategy has less messaging overhead while it is plausible that it could still be a good estimator for the optimal connection. We therefore include it in our experiments.

The triggers we will evaluate are those described in Section 2.3. The scheduling trigger will however not be evaluated, as its implementation is considered too complicated and unwieldy for this project. This leaves four triggers to be examined:

- Optimal
- Threshold
- Hysteresis
- Hysteresis and threshold

Combining the two metrics and four triggers gives us eight data handover strategies to be implemented and evaluated.

4.4 Classes of applications

It would be imprudent to assume that there is a single optimal strategy for all different applications. After all, the properties of applications can vary wildly. For our research in MEC, the properties that we consider to have the most influence over what makes an optimal handover strategy are the following:

- *Service rate*, the frequency with which the application makes requests to the server.
- *Service duration*, the amount of processing required from the server for a single service request.
- *Service message size*, the size of the service request and service response messages.
- *Handover message size*, the size of the message sent from the old server to the new server upon UE handover.
- *Delay tolerance*, the maximum acceptable RTT on a service request from a UE.

Each MEC application has an individual set of these properties, which can affect what the best strategy is for that application.

For example, an application that sends requests to the server at a high frequency might have multiple requests suffering from the connectionless period during a handover, while an application that sends with lower frequency will only have one or even no messages interrupted. This might cause the former to prefer a strategy with fewer handovers, while the latter will not be penalized as strongly and might benefit from a more aggressive handover strategy. Similarly, an application that has processing-heavy jobs will be less heavily effected by queuing time than an application that has low processing requirements. An application with a high delay tolerance, for example, because it has a built-in buffering strategy, might settle for a sub-optimal server selection to avoid frequent handovers, while an application with extremely low delay tolerance, for example a safety-critical application, will benefit from a more aggressive handover strategy to reap the benefits of a slightly lower RTT.

As discussed in Section 4.2, this work considers a use case that is stateless and therefore transfers little data upon handover. Furthermore, it has a very low delay tolerance and a high service rate, while message sizes are relatively small. We expect that for an application that has different properties, the optimal data handover strategy will be different than in our results. We suggest that this be tested in the future. To this end, the simulation system was built in such a way that it is very easy to configure for another application type.

Chapter 5

Implementation

To execute the experiments as detailed in the previous chapter, an implementation was made in ns-3. The code can be found on GitHub: [21]. This chapter details exactly how it was implemented. It covers the main actors in the system, as well as an in-depth explanation of the implemented processes. Finally, it discusses the topology of the network used for the simulations, as well as a complete overview of the experiment parameters.

5.1 Actors

The implemented system consists of three main types of actors; MEC servers, UEs, and a single orchestrator. The functions and responsibilities of each are outlined in the following sections.

5.1.1 UE

The system's main actors are the UEs. Each UE is on board of a separate vehicle, and has the client role in most interactions. It requests service from the MEC server it is connected to and measures the RTT. Each UE has a unique mobility pattern associated with it so that no two UEs will have identical mobility profiles. The number of UEs in the system can be easily adjusted to fit extended experiments.

5.1.2 MEC server

The MEC server provides service to the UEs that are connected to it. It also communicates with the orchestrator to execute data handovers from one MEC server to another. It is possible to alter the number of MEC servers in the system, although the process is slightly more involved than it is for UEs.

MEC servers are governed by two experiment parameters; the combined server capacity and the server capacity distribution. The former specifies how many jobs all MEC servers combined can process per (milli)second. The latter specifies how this computational capacity is distributed among the MEC servers. For example, a [0.33,0.33,0.33]-distribution implies that all three servers in the system get an equal share of the total server capacity, while a [0.4, 0.5, 0.1]-distribution means that when the servers have an equal amount of clients each, waiting times in the first server will rise more than in the second server. Note that even if the combined server capacity far exceeds what the system generates, a poor handover strategy may cause individual servers to be overloaded.

5.1.3 Orchestrator

The orchestrator is the actor that makes the decision of when and where a UE should do a data handover. There is exactly one in the system. It receives measurements of the system state as taken by the UEs and MEC servers and combines these with the selected data handover strategy. If a handover is to be made, the orchestrator sends the appropriate instructions to the involved parties. A more detailed description of this process can be found in Section 5.3.3. Important experiment parameters for this actor are the threshold and hysteresis values. For delay-based strategies, the threshold is given in ms; if the UE's measured RTT exceeds that, an alternate MEC server will be sought.

For distance-based strategies the threshold is set in meters. The hysteresis parameter determines what percentage of performance increase another MEC server must offer before the UE will consider switching to it. In the experiments, we use a hysteresis of 15%. Pre-experiments found that this value deters route-flapping, but does not completely deter handovers by setting an unachievable standard.

5.2 Topology

To connect all the actors and enable them to run their applications, a network was implemented in ns-3. This section details the design for that network, including the major parameter settings that were chosen.

The full topology of the network can be seen in Figure 5.1. In our experiments, there are 100 UEs, three eNBs and their associated MEC servers, and an orchestrator. It also contains a packet gateway (PGW) and an IP router.

It was found that creating a true MEC implementation in ns-3, that is, an implementation where the servers are connected directly to an eNB, is prohibitively complicated. We have therefore chosen to approximate MEC by making the servers accessible through the core network, and setting network delays in the core network (between the PGW and the router, as well as between the router and each server) to zero. In a true MEC implementation, a UE could circumvent the core network by connecting to the server associated with their current eNB. In our implementation, the core network does not impose any delay. However, a 15 ms penalty is incurred whenever a UE connects to a server that is not associated with their current eNB, thereby simulating the core network delay. This way, the implementation is functionally the same as a true MEC implementation and the experiments will provide realistic results.

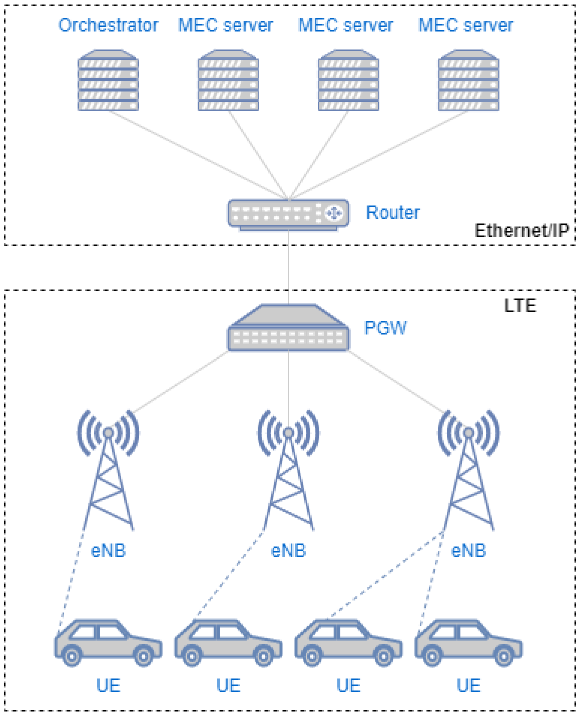

Figure 5.1: *Network topology for the implemented system*

5.3 Processes

The implemented system consists of three processes; service requesting, status reporting and handover. The following sections will describe for each process their purpose, the responsibilities of each actor, and the control flow for this process.

5.3.1 Service requesting

The service request process is the core of the system. This process implements the regular interaction between a UE and a MEC server. The UE sends a service request at a certain interval, to which the MEC server then replies. The request interval, size of the request message, and size of the response message are all parameters that can be set. The UE measures the RTT for each service request and logs it. This data is used to complete an analysis that will determine whether a data handover is necessary.

The control flow, which can be seen in Figure

5.2, is as follows; the process is instigated by the UE through an interval timer. The UE sends a request message to its MEC server and starts the RTT timer. The MEC server receives the message and adds it to the processing queue. Based on the queue's current length, the server calculates when the message will be done processing. The server sets a timer, and when the waiting time has elapsed it sends a response to the UE. The UE receives the response, stops the RTT timer, and logs the simulation time and delay. A new trigger is set for the next service request.

To prevent all UEs from sending their service requests at the same time and creating bursty traffic, the service request triggers are staggered. If each UE would send a request at the exact same time, it would lead to short periods of the servers being overwhelmed before returning to a stable queue state. This would cause sub-optimal performance. When staggering the request, each UE is allotted a narrow time slot in which to send their messages, creating a more uniform input flow for the servers and thereby improving performance. This is also a more realistic scenario; while in the simulation all vehicles are perfectly time-synchronized, it is very unlikely that this would be true in the real world.

5.3.2 Status reporting

Status reporting is one of the support processes in the system. It does not provide services to the UEs or MEC servers directly but is required to help the system function. In this process, the UEs and MEC servers periodically update the orchestrator regarding their perceived system status. The process consists of separate parts for UEs and MEC servers.

The status reporting for UEs is dependent on the metric that is being used. If the applied metric is delay, each UE periodically updates the orchestrator about their RTT to each of the servers. Refer to Figure 5.3 for the activity diagram. For readability, the diagram only shows the UEs communication with a single MEC server. By default, the UE will only know the RTT to the server it is currently connected to, by its measurement of the last service request. It has no information on any of the other servers in the system. To solve this, each UE sends a ping request to *each* of the servers in the system. This process is triggered by the ping timer running out. The timer is configurable and is separate from the service timer. A ping request is almost identical to a regular service request; it has the same size and is handled in the same way by a MEC server. However, it has a flag set so that the UE will recognize it as a ping response upon return. The UE sends each request and sets separate RTT timers. When the MEC servers receive the requests, they handle it the same as a service request; they calculate the waiting time based on current queue length, set a timer, and send a response when the timer runs out. Once all the ping responses have been received, the UE bundles the gathered information and sends a message called a measurement report to the orchestrator. Finally, the timer is reset.

When the distance metric is in use, the UE does not inform the orchestrator of the RTTs, but its current position. This makes the process significantly simpler; when the ping timer runs out, the UE simply sends a message containing the current position for the UE to the orchestrator. Then the ping timer is reset and the process starts over.

Each MEC server also regularly updates the orchestrator, triggered by a third timer called the server timer. Once again, this timer is configured in the configuration file and is separate from the other timers. When the timer runs out, the MEC calculates its current waiting time and sends a message containing this value to the orchestrator. Finally, the timer is reset.

5.3.3 Data handover

The final process that is running in the system is the handover process. In this process, the orchestrator reviews the data it has received from the UEs and MEC servers, and decides whether or not a data handover should take place. If a handover should be made, the orchestrator sends commands to the parties involved; the relevant UE and its current MEC server. If no handover needs to take place, the orchestrator does not send any messages. It would have been possible to implement this functionality in the

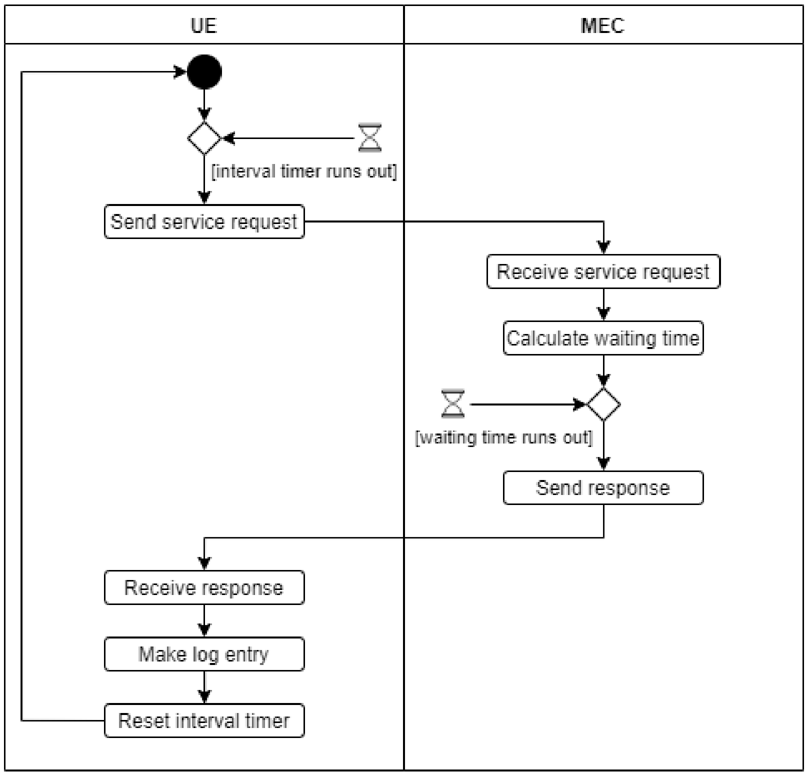

Figure 5.2: *Activity diagram of the service request process*

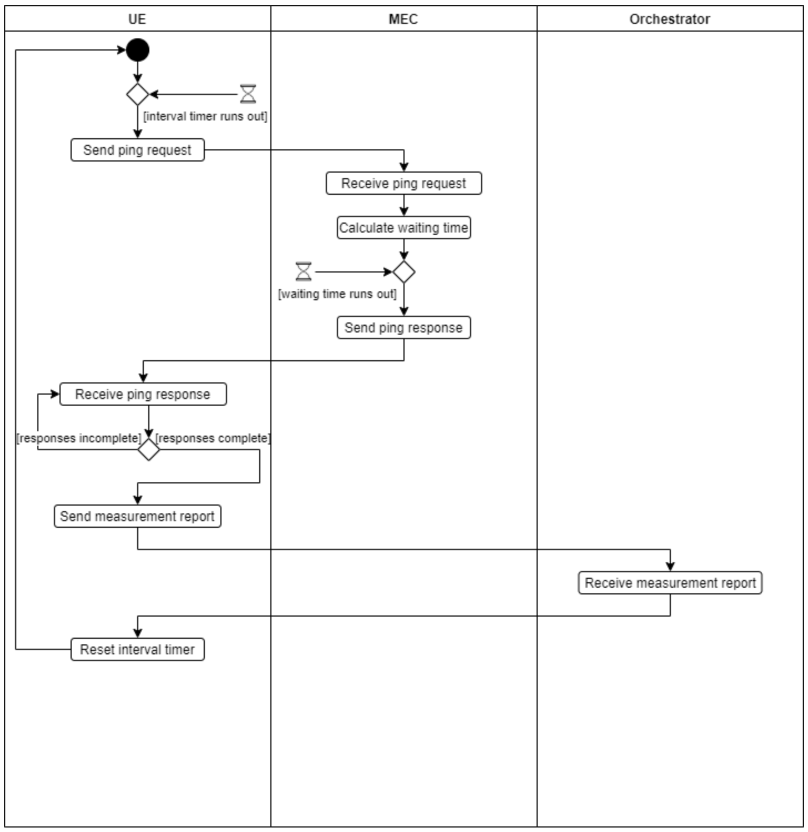

Figure 5.3: *Activity diagram of the UE status reporting process for delay-metric strategies*

UEs themselves rather than in a single, separate entity; however we made this decision to have a clearer separation of concerns between the various system components.

The control flow, visualized in Figure 5.5, begins with the orchestrator receiving a measurement report containing either the RTTs for all MECs or the UEs current position. The orchestrator then uses these numbers to fill out one of the data handover equations; which one is used depends on the handover strategy that is set in the configuration file. The data handover equations depend on the trigger that is used and can be found in Table 5.4. Here, the variables *current* and *other* represent the numerical values for the metric for the UEs current MEC server and the MEC server it is being compared to, respectively. The RTT is measured in milliseconds and distance is measured in meters. The variables threshold and hysteresis are fractions defined in the configuration file. They are kept equal between experiments.

If at least one of the MEC servers that the UE is not connected to triggers the handover condition, a data handover is initiated to the most favourable MEC server.

Our simulator, like LTE does, uses hard handover. This means that the connection to the old server is severed before the connection to the new one is established. As a consequence, there is a short period in which the UE is unable to send requests to *any* server. We call this the no-send period. The no-send period lasts until both MEC servers have processed the handover request. Should the UE need to send a service request during the no-send period, the RTT timer is started, but the message will not be transmitted until the no-send period is over, leading to an increased RTT during this time. As ns-3 does not know this concept, the no-send period is calculated by the application when a data handover is initiated.

When a data handover is initiated, the first step is for the orchestrator to calculate the no-send period for the UE based on the response time at its current MEC server and the MEC server it is being handed over to. Then, the orchestrator sends the command to hand over to the UE, and then sends a similar command to its current MEC server. MEC servers do not disconnect from the infrastructure when a UE hands over and therefore do not have to abide by the no-send period. Once the current MEC server receives the command to hand over, it transmits a message to the new MEC server to inform it of its new client. The size of this message can be configured, allowing the system to mimic applications that have to transmit varying amounts of user data in a handover. The old MEC server then removes the UE from its client list, while the new MEC server adds the UE to theirs. The data handover is now complete.

5.3.4 Experiment parameters

An overview of all parameters that can be set, as well as the values we have used in our experiments can be found in Table 5.6.

Note that the total server capacity is calculated as follows: each vehicle produces 11 messages for a server to process per second; 10 service requests + 1 ping request. In a 100-vehicle scenario, this means that in order to be stable, the server capacity must be at least 1100 jobs/second. To ensure there is not too much queuing influencing our results, we chose to have double the minimum needed capacity; 2200 jobs/second.

Trigger	**Equation**
Optimal	$other < current$
Threshold	$other < current$ && $current > threshold$
Hysteresis	$other < current * (1 - hysteresis)$
Hysteresis & threshold	$other < current * (1 - hysteresis)$ && $current > threshold$

Figure 5.4: *Overview of data handover equation*

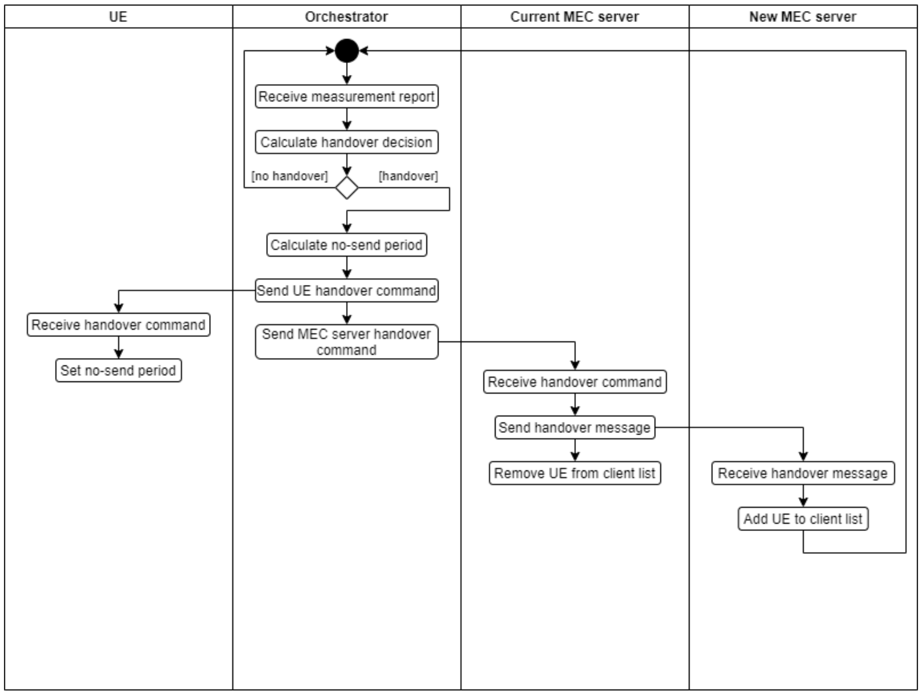

Figure 5.5: *Activity diagram of the data handover decision and execution process*

Parameter	**Value**	**Reasoning**
Number of eNBs	3	Multiple handover opportunities, not overly complex.
Number of servers	3	One per eNB.
Number of UEs	100	Modest system load with feasible experiment duration.
Distance between eNBs	4000 m	Even spacing between eNBs, mimicking TNO test site.
Simulation runtime	1820 s	Time for each UE to do at least one full lap.
Service request packet size	256 bytes	Realistic value for TNO platooning application.
Service response packet size	256 bytes	Realistic value for TNO platooning application.
Handover UE data packet size	256 bytes	Realistic value for TNO platooning application.
Handover command packet size	256 bytes	Realistic value for TNO platooning application.
Ping interval	1000 ms	Avoids influencing system load-based measurements.
Position update interval	1000 ms	Avoids influencing system load-based measurements.
Service request interval	100 ms	Realistic value for TNO platooning application.
Total server capacity	2200 jobs/second	Creates some queuing at servers, but no systematic overload.
Server capacity distribution	[0.4,0.5,0.1]	Prevents closest server preference in delay metrics.
Hysteresis	15%	Anti-route flapping, still achievable.
Threshold (delay)	100 ms	Anti-route flapping, limits performance degradation.
Threshold (distance)	2200 m	10 % more than optimal.
Vehicle maximum speed	27.8 m/s	Mimics TNO test site.
Vehicle maximum acceleration	3.5 m/s^2	SUMO recommended value.
Vehicle maximum deceleration	2.2 m/s^2	SUMO recommended value.
Car-following model	Krauss	SUMO recommended value.
Lane-changing model	SUMO default	SUMO recommended value.

Figure 5.6: *Overview of simulator parameters*

Chapter 6

Results

This chapter discusses the experiment results. All results are based on 10 replications of each experiment in the simulator, each with their own mobility file and random generator seed. For ease of reading, the chapter is split up into four sections; one that discusses the handover frequency affected by each strategy, one that discusses the number of UEs connected to each server, one that covers the round trip times measured by the UEs, and finally a section describing the RTT violations per strategy. Recall from Section 4.1 that we consider the likelihood of RTT violations to be the decisive factor for an optimal strategy; the other sections are included for clarification of the simulator's behavior.

The results discussed in this section are those that are measured in a steady-state; the initial transient has been removed from the results. The initial transient takes place in the approximately 1000 seconds before the vehicles are spread evenly along the track. In the results, we start counting experiment time from the moment that steady-state is achieved.

6.1 Handover frequency

We start our results with a look at the system-wide handover frequency in each strategy. This indicates how many of the UEs perform a data handover in a given second of the simulation. Graph 6.1 plots the handovers per second versus the simulation time in seconds. A second graph, Graph 6.2, has been added to display more clearly the lower-frequency strategies. From these graphs, we can clearly see a number of things. First of all, the delay-optimal strategy gives rise to the highest handover frequencies. This is expected; with the optimal trigger, a data handover will be performed even if the alternative server is only minimally better than the current one, while both the hysteresis and threshold triggers will not handover until there is a sufficient gain from doing so. We can see that the delay-optimal strategy creates route flapping, where the delay-hysteresis strategy significantly reduces this effect. However, the delay-threshold strategy is even more efficient at this. We can also see that the delay-threshold & hysteresis strategy more closely follows the line of the threshold only trigger than it does the hysteresis-only trigger. This indicates that the threshold (set at 100 ms) has a much more significant effect on the handover decision than the hysteresis parameter does, demonstrating that if the threshold is met, there is almost always an alternative that is at least 15% better than the current server, but the reverse is not true.

At approximately 320 seconds into the experiment, we see that two of the strategies, delay-optimal and delay-hysteresis, suddenly have a large increase in handover frequency. The handover frequency graphs cannot provide an explanation for this behavior, but graphs of the individual servers' response times, the time between the arrival of a message in the server and the departure of the corresponding response, provide us an explanation. Graph 6.3 shows the response times of the servers in the delay-optimal strategy, while Graph 6.4 shows the same for the delay-hysteresis strategy. In both graphs, we see that server 3, which has the least computational capacity of all servers, becomes overloaded. The response time shoots up and

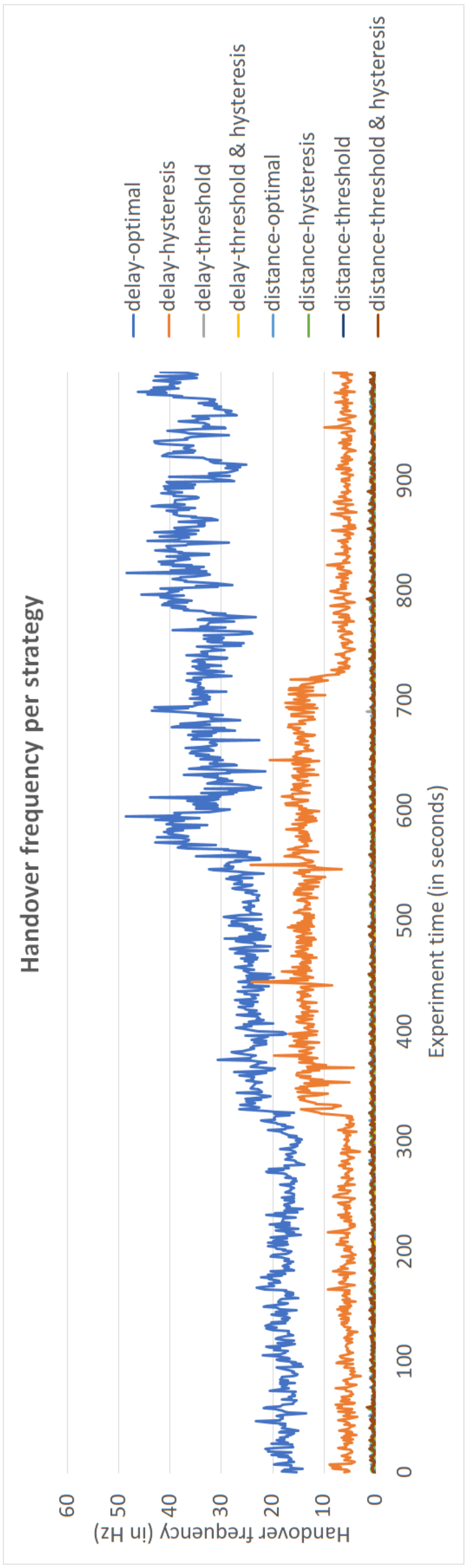

Figure 6.1: *Handover frequency per strategy*

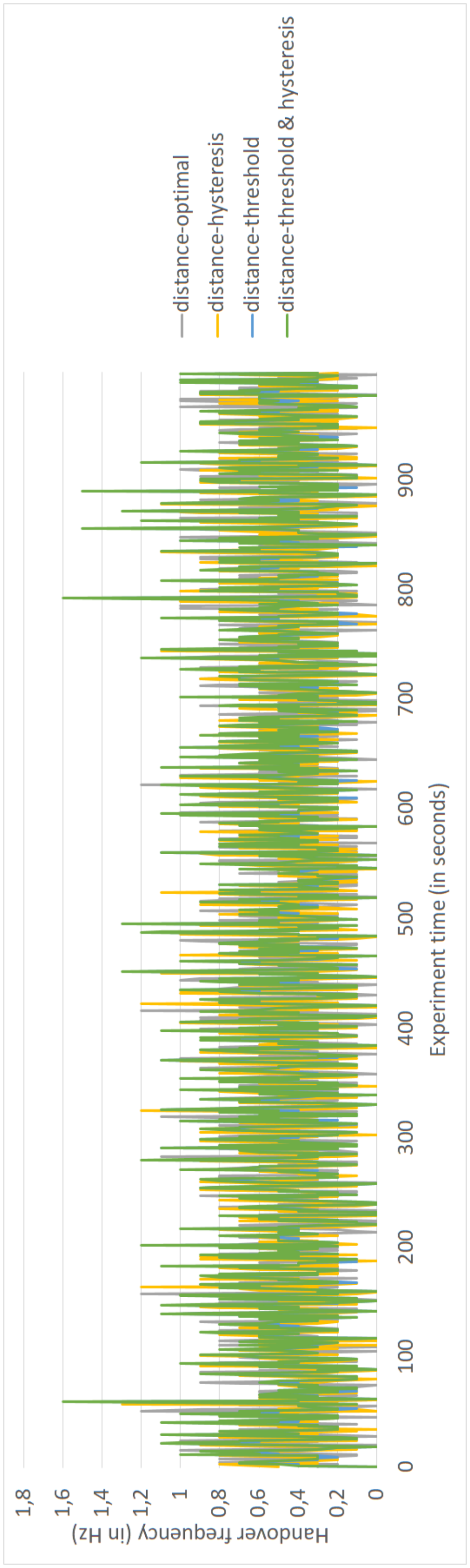

Figure 6.2: *Handover frequency in low handover frequency strategies*

for both strategies the UEs connected to this server will leave to find better connectivity at servers 1 or 2. This causes the response time of server 3 to plummet, making it once again a viable candidate, and it quickly regains so many clients that it becomes overwhelmed again. The opportunistic nature of the delay-optimal strategy causes the response time to remain imbalanced for the duration of the experiment, while the hysteresis trigger clearly provides a dampening effect, eventually causing the server response times to return to a mostly steady-state. The strong fluctuation in server response times creates a flood of data handovers in either situation, which we can observe in the blue and orange lines of Figure 6.1.

The graphs also show that the distance-based strategies behave very differently than the delay-based ones, as is best illustrated in Figure 6.2. As expected, they perform handovers much less frequently; a UEs position is much more stable than the response time it is measuring. The figure also reveals that the triggers do affect the handover decision, but not nearly as strongly as they do in the delay-based scenarios; the different entries have very similar, though not equal trends. This is caused by the fact that handover decisions in distance-based strategies are based on the position of the car, and as the same mobility seeds are used for each set of experiments, the vehicles' mobility is identical in each scenario. Due to the experiment parameters the exact point of handover is not the same between strategies. As servers are 4000 meters apart, the optimal trigger will cause a handover as a vehicle passes the 2000 meter mark between servers, but the hysteresis trigger will not perform a handover until there is an alternative server that is at least 15% closer than its current provider, meaning the handover will happen a few seconds later in comparison. As the measurement of distance to a server is as-the-crow-flies, the span of time between when the optimal strategy hands over and when a strategy based on another trigger will can be slightly different when on a straight stretch of road vs. in a turn. This explains why the hysteresis, threshold and hysteresis & threshold strategies do not yield a linear transposition of the optimal strategy, though they are very similar.

Figure 6.5 shows the average frequency of handovers per strategy. The vertical lines on the bars indicate the 95% confidence intervals. Here we can see very clearly that for the distance-metric strategies, the trigger plays a very minimal role in determining the number of data handovers a UE performs. Furthermore, the figure illustrates the large difference in handover frequencies between the various delay-metric strategies.

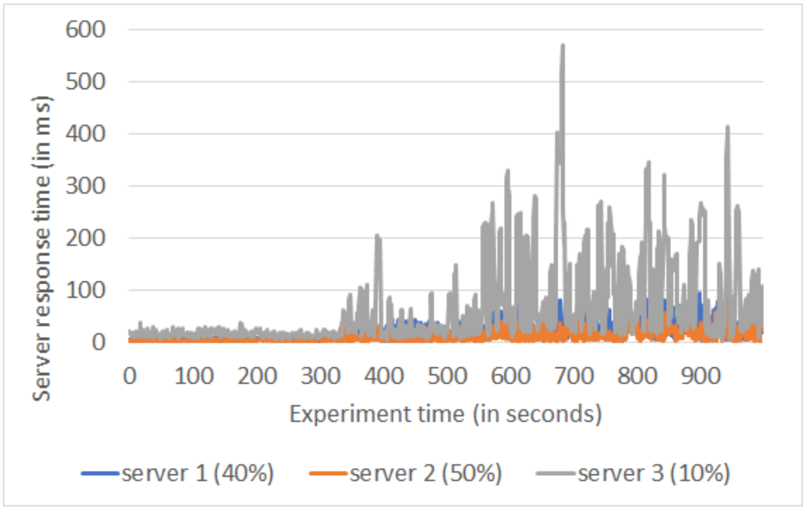

Figure 6.3: *Server response times for delay-optimal strategy*

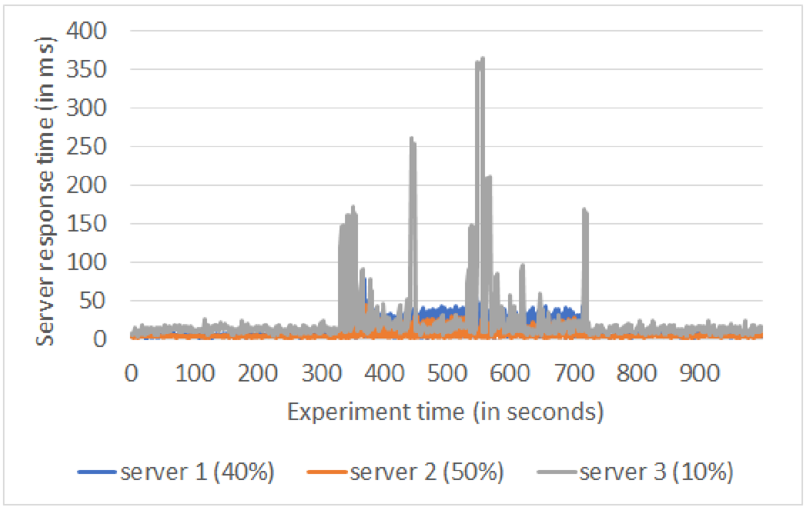

Figure 6.4: *Server response times for delay-hysteresis strategy*

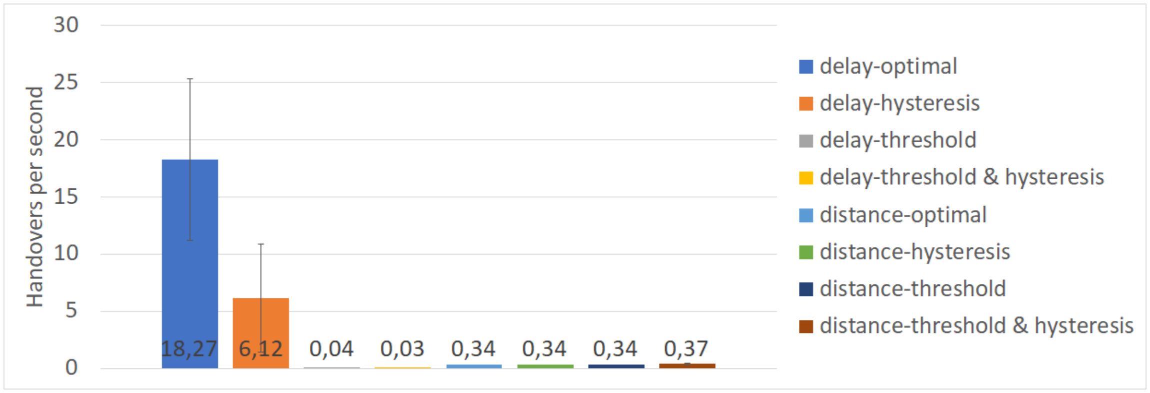

Figure 6.5: *Average handover frequency & confidence interval per strategy*

6.2 Clients per server

In this section, we will discuss the distribution of UEs over the various servers. Figures 6.6 through 6.13 show the number of clients per server over time for the eight strategies. Refer to Figure 6.6 for delay-optimal, Figure 6.7 for delay-hysteresis, Figure 6.8 for delay-threshold and Figure 6.9 for delay-threshold & hysteresis strategy graphs. In a similar order, Figure 6.10 shows the distance-optimal, Figure 6.11 distance-hysteresis, Figure 6.12 distance-threshold and Figure 6.13 the distance-threshold & hysteresis strategy.

We can see that the strategies that hand over most frequently, seen in 6.6 and 6.7, that the frequent handovers result in a more unbalanced system compared to strategies where handovers are less frequent, such as the delay-threshold and delay-hysteresis & threshold strategies.

We can also see that in the delay-metric strategies, server three, the weakest server in the system, remains slightly underutilized. It generally does not handle more than about 5% of the populace, while it holds 10% of the processing power. This indicates that server 3 is generally not capable of providing better service than the other servers are. This effect is exacerbated when the threshold or hysteresis triggers are applied, as can be seen from figures 6.7, 6.8 and 6.9. These put an even stricter expectation on an alternative server, and we can observe server 3 being even further underutilized as a result.

Figures 6.10 through 6.13 show us that the distance-metric strategies are identical to one another in this performance aspect. This is explained by the fact that which server a UE connects to is determined by its mobility, which does not change between the experiments. Also observe that in these scenarios, each server has approximately the same number of clients, which indicates that the vehicles are indeed spread evenly along the track.

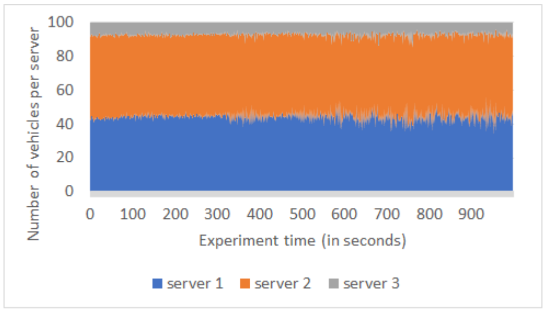

Figure 6.6: *Number of clients per server, delay-optimal strategy*

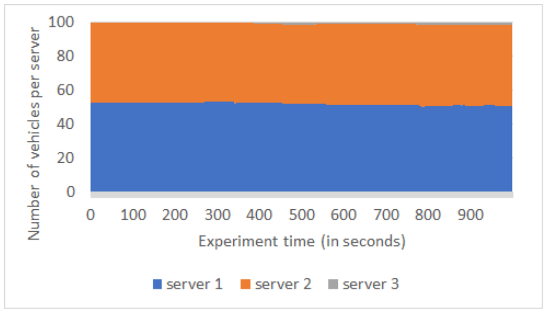

Figure 6.9: *Number of clients per server, delay-threshold & hysteresis strategy*

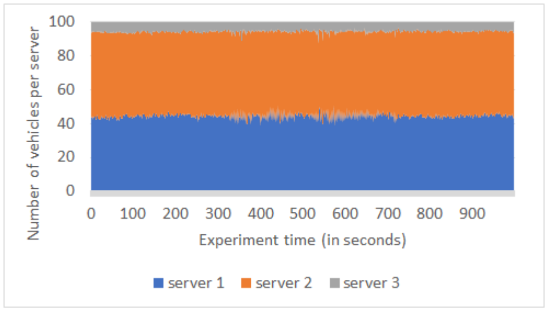

Figure 6.7: *Number of clients per server, delay-hysteresis strategy*

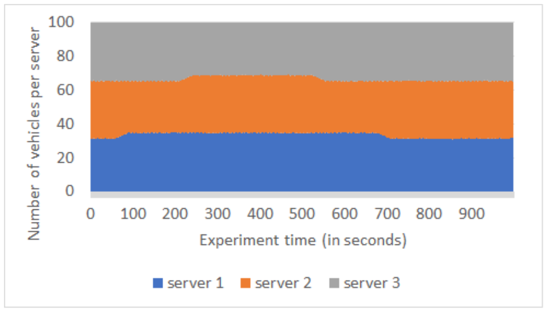

Figure 6.10: *Number of clients per server, distance-optimal strategy*

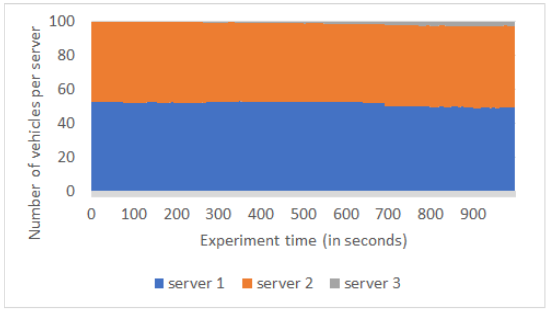

Figure 6.8: *Number of clients per server, delay-threshold strategy*

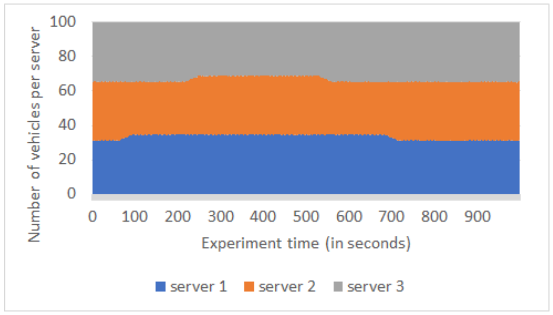

Figure 6.11: *Number of clients per server, distance-hysteresis strategy*

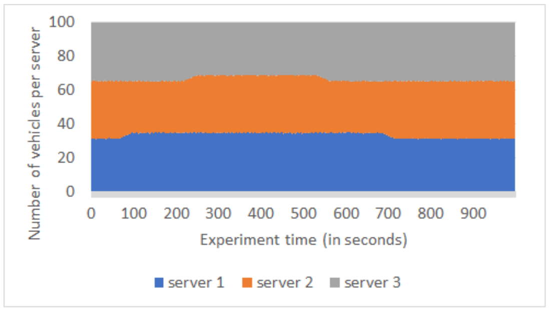

Figure 6.12: *Number of clients per server, distance-threshold strategy*

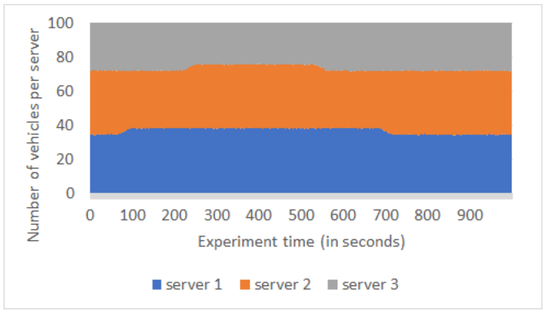

Figure 6.13: *Number of clients per server, distance-threshold & hysteresis strategy*

6.3 RTT

This section discusses the (average) RTT as observed by the UEs. Figures 6.14 and 6.15 show the average RTT UEs experienced over time for delay-metric strategies and distance-metric strategies, respectively. Figures 6.17 and 6.18 condense this into an overall average. These graphs also show error bars; these represent the edges of the 95% confidence interval for the RTT.

From the aforementioned graphs, we can see that the delay-metric strategies by far outperform the distance-metric strategies (note the different scales in figures 6.14 and 6.15). This can be easily explained; in a delay-metric strategy, if a UE finds its current server overloaded, it will move to a more suitable one. In the distance-metric strategies, this escape is not possible, and the problem continues to escalate until the UE travels far enough to connect to another server. Even though vehicles are spread evenly across the track, servers can still get overloaded in distance-metric strategies, because they are not by definition capable of shouldering a fair part of the workload.

Figure 6.17 shows that the optimal trigger results in the highest average RTT of all delay-metric strategies. Figure 6.14 reveals that it is also the most variable of all strategies, followed by the hysteresis trigger. As discussed in Section 6.1, the instability is triggered by the weakest of the three servers in the system getting overloaded. The optimal and hysteresis triggers are satisfied easily, causing route flapping. The resulting high number of handovers degrades system performance and causes high and fluctuating RTTs. The delay-hysteresis strategy eventually stabilizes because its trigger somewhat delays handovers, but the delay-optimal strategy continues to fluctuate wildly for the remainder of the experiments.

For delay-metric strategies, the threshold and threshold & hysteresis triggers perform best. They perform so similarly that the line representing the delay-threshold strategy entirely disappears behind the delay-threshold & hysteresis line. Figure 6.16 provides a detailed view of just how small the differences are. This indicates that the delay-threshold & hysteresis strategy's behavior is vastly more influenced by the threshold requirement than the hysteresis. Apparently, there is almost always an alternative server that provides a 15% better RTT when the current server's RTT has degraded to more than 100 ms.

For the distance-metric strategies shown in Figure 6.15, we see that the results are nearly identical. As in the previous sections, this can be explained by the fact that with a distance metric, the triggers do not influence how often or which handover decision is made, but only the timing. Because the vehicles' mobility is identical for each set of experiments, there are only minor time-shifts between the strategies.

Observe from Figure 6.18 that there is some difference between the different distance-metric strategies. However, we do not consider the difference to be statistically significant, as the 95% confidence intervals (marked by the error bars) largely overlap.

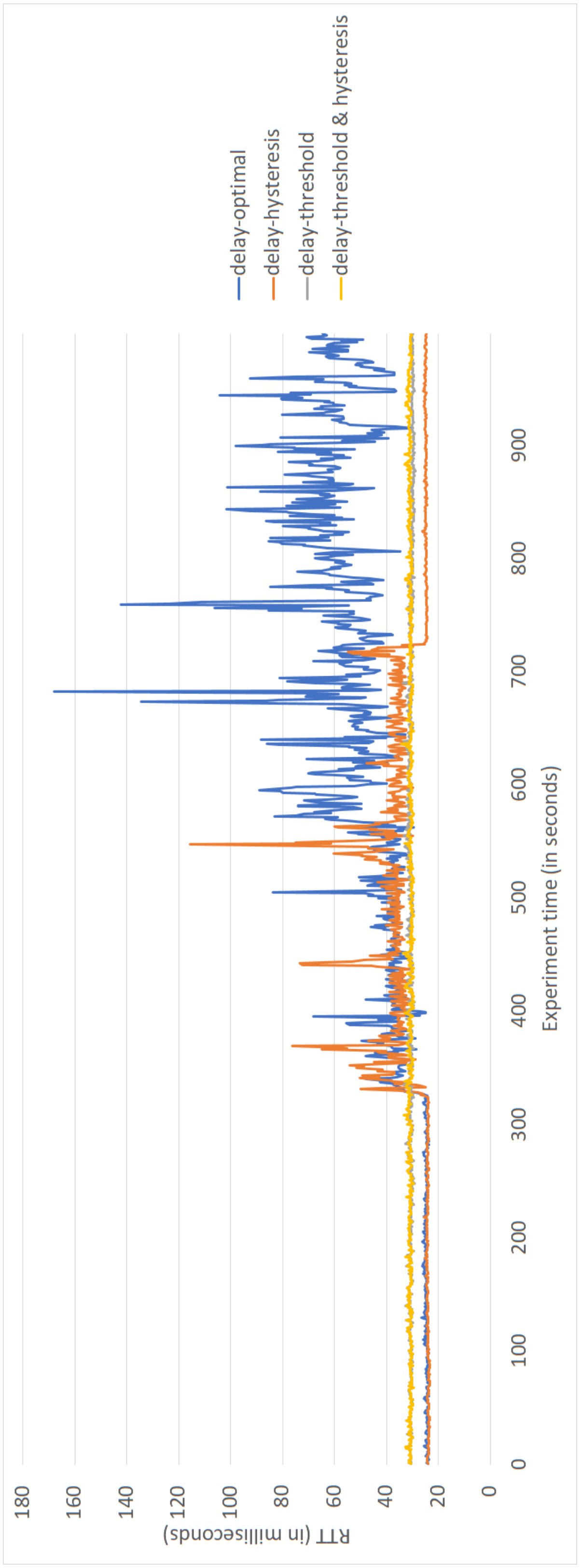

Figure 6.14: *Average RTT in delay-metric strategies*

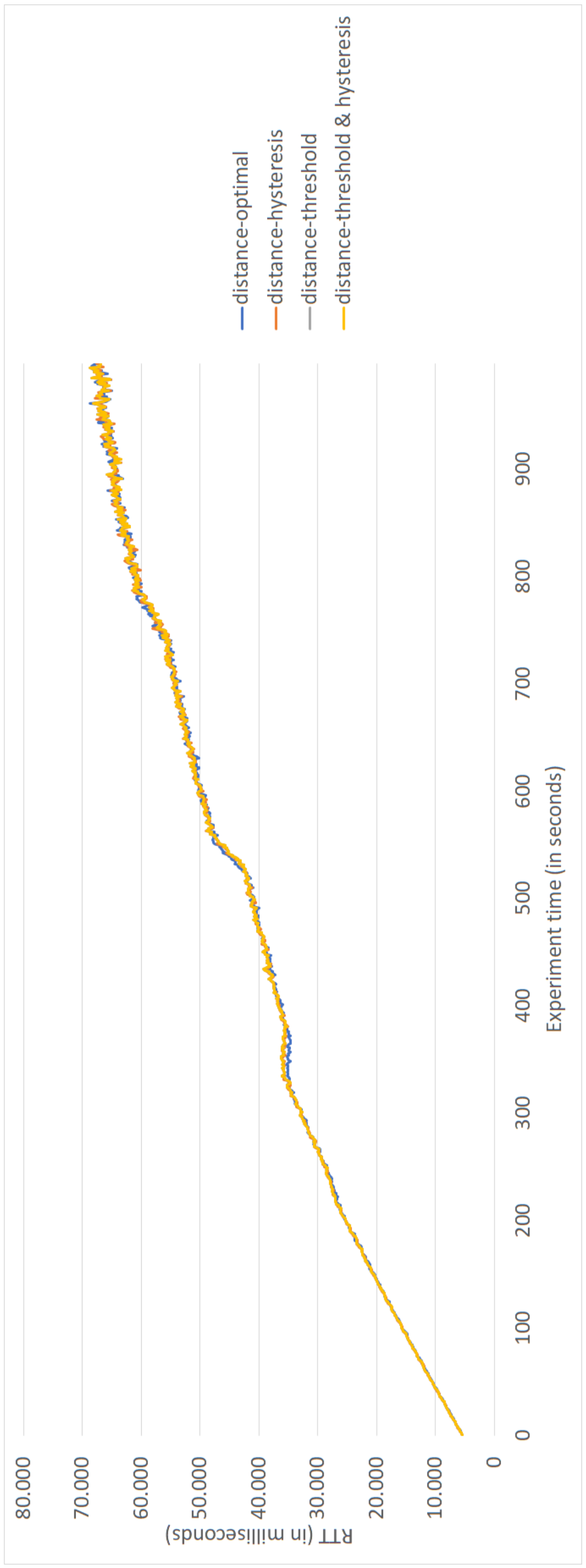

Figure 6.15: *Average RTT in distance-metric strategies*

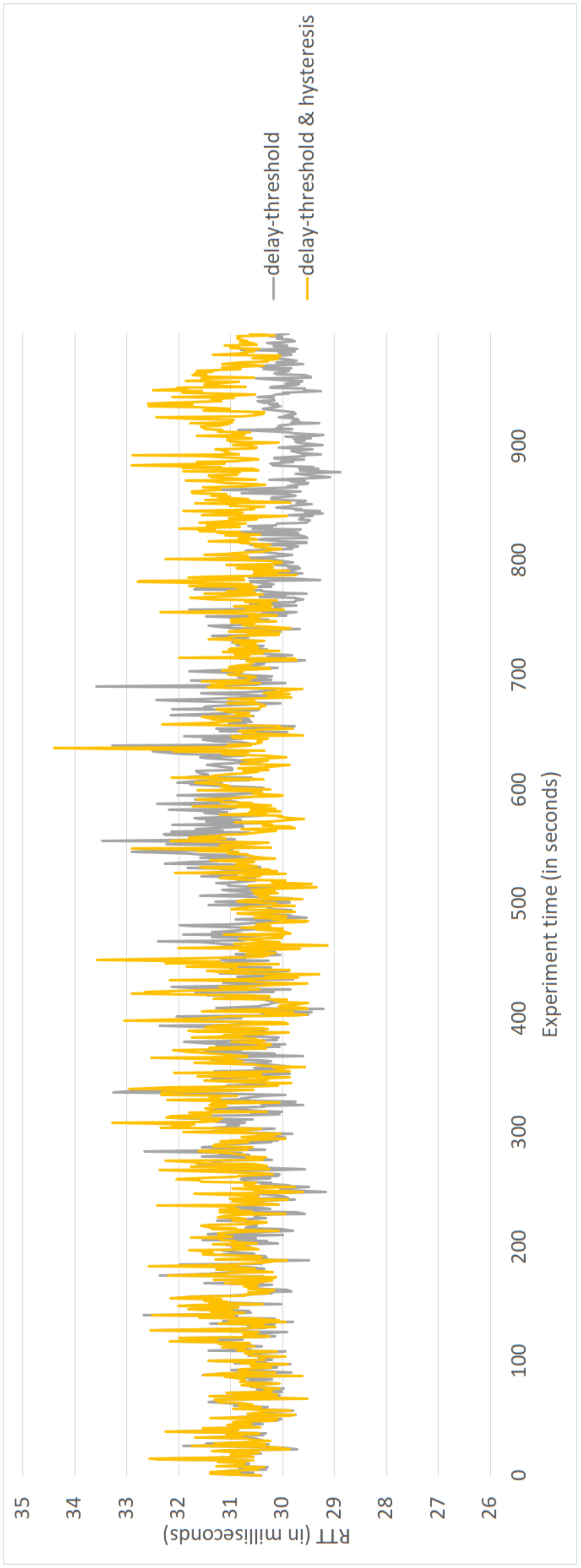

Figure 6.16: *Average RTT in delay-threshold & delay-threshold & hysteresis strategies*

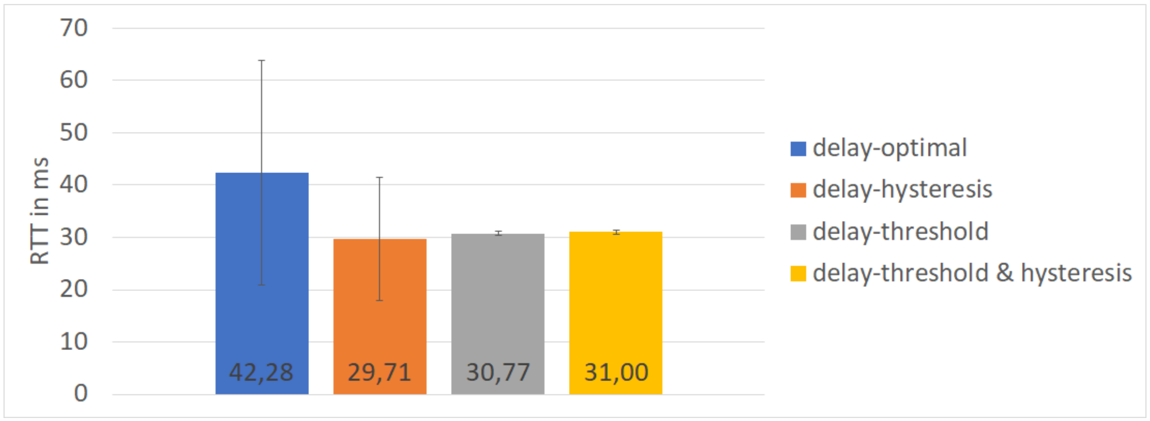

Figure 6.17: *Average RTTs, delay-metric strategies*

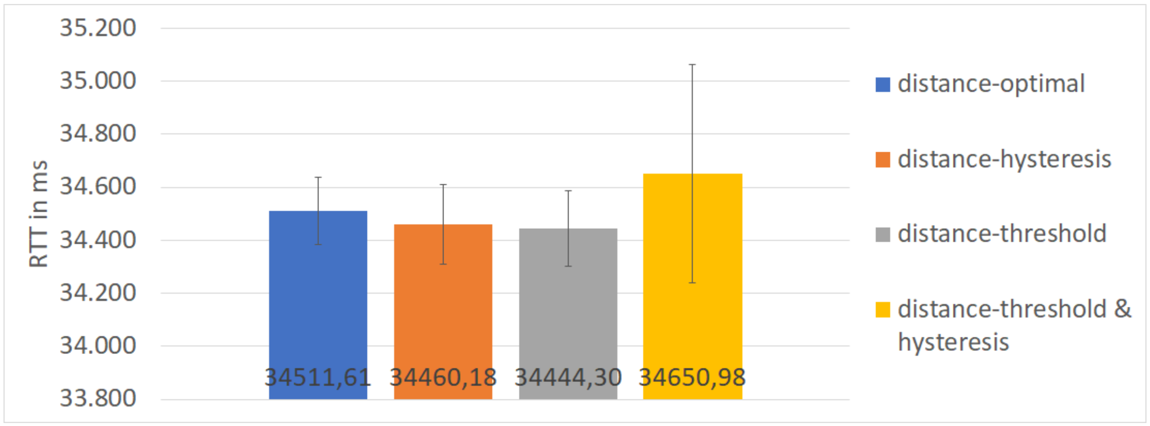

Figure 6.18: *Average RTTs, distance-metric strategies*

6.4 RTT violations

As the likelihood of RTT violations is the metric by which we decide which is the optimal strategy for the TNO platooning application, this is the most important section of our results. In our experiments, we aggregate all the RTT measurements a UE does in a second, and log the average of these. We refer to these as 'moments'; if in a given second the average RTT exceeds 30 ms the moment violates the RTT constraints, and if it does not, the moment is in time. This choice was made to decrease the time it takes to run an experiment, as ns-3 simulations are slow, and writing to file is one of the more time-intensive operations. This way of measuring is minimally less accurate as we are using averages rather than separate values. However, at 100 vehicles per experiment and each experiment being replicated 10 times, we have 1000 points of measurement per moment, which minimizes the effect while allowing us to run longer experiments.

Figure 6.19 shows us the total number of in-time moments (in green) and the number of moments that violate the RTT requirements (in orange). We can see that there are significant differences between the strategies; the delay-optimal strategy is able to deliver over 25.000 more in-time moments than the delay-threshold strategy, for example. At first glance, the distance-metric strategies also seem to perform better than one might expect from the RTT results. However, in this case, appearances are deceiving; note that the total number of moments for the distance strategies is much lower than they are for their delay-metric counterparts. As we know that the vehicles are driving for the same amount of time in each scenario, this indicates that there are moments in which it is impossible to determine an average for that moment. This happens when the RTT is so high that a UE does not receive any responses in a given moment. We can therefore state that these moments also violate the RTT requirements.

It is better to examine the percentage of in-time moments versus the total moments. This overview is provided in Figure 6.20. From this Figure we can see that the delay-optimal and delay-hysteresis strategies have a much higher rate of successful deliveries than other strategies. This may seem counter-intuitive, as we have seen in Section 6.3 that the RTT for these strategies becomes and remains unstable once one of the MEC servers has become overloaded. However, the high peaks are interspersed with moments of low RTT where the deadlines *are* met. The delay-threshold and delay-threshold & hysteresis are much more stable, but as can be seen in Figure 6.17, they remain stable just above the required 30 ms RTT. Therefore, it pays off in this scenario to be opportunistic, even though it also means some high peaks. The alternative is to almost always miss the deadline. However, if an application is slightly more delay-tolerant, that opportunism will not be worthwhile any longer.

Another behaviour that may seem surprising based on the previously shown results; the distance-metric strategies perform fairly well compared to the delay-metric ones, and in some cases even easily outperform them. This seems strange, when in Section 6.3 we saw that the average RTT increases dramatically for these strategies over the duration of the experiment. Figure 6.21, showing the average response time per server for each strategy, provides insight as to the cause of this behavior; while the server with the lowest processing power becomes overloaded and response times skyrocket, the other servers can handle their traffic and response times stay remarkably low. As the vehicles are spread equally among the servers, this means that two-thirds of the vehicles are connected to a server that has a chance of delivering messages in time. These results imply that in a scenario where each server is powerful enough to handle their portion of the traffic, distance-metric strategies may be a better choice than delay-metric ones.

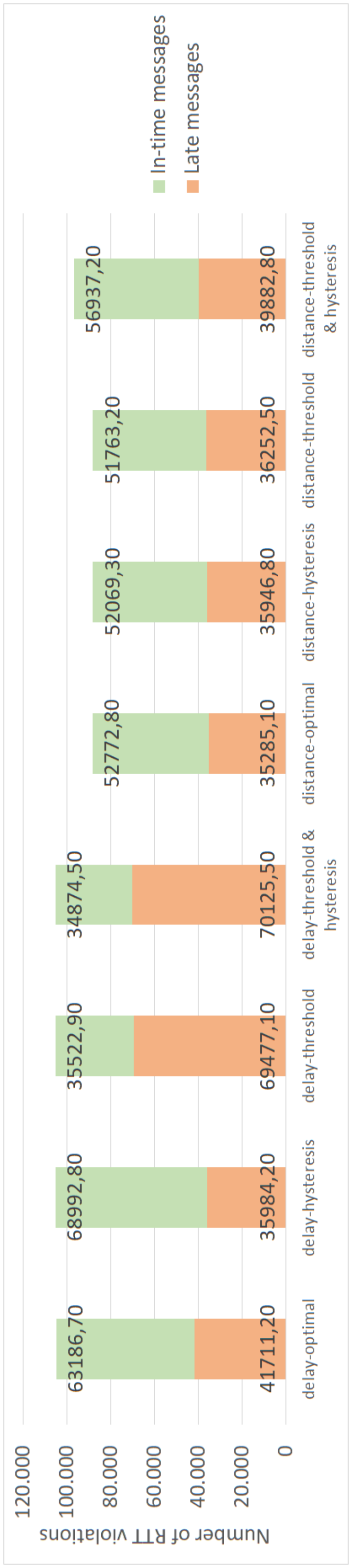

Figure 6.19: *Number of moments with RTT violations*

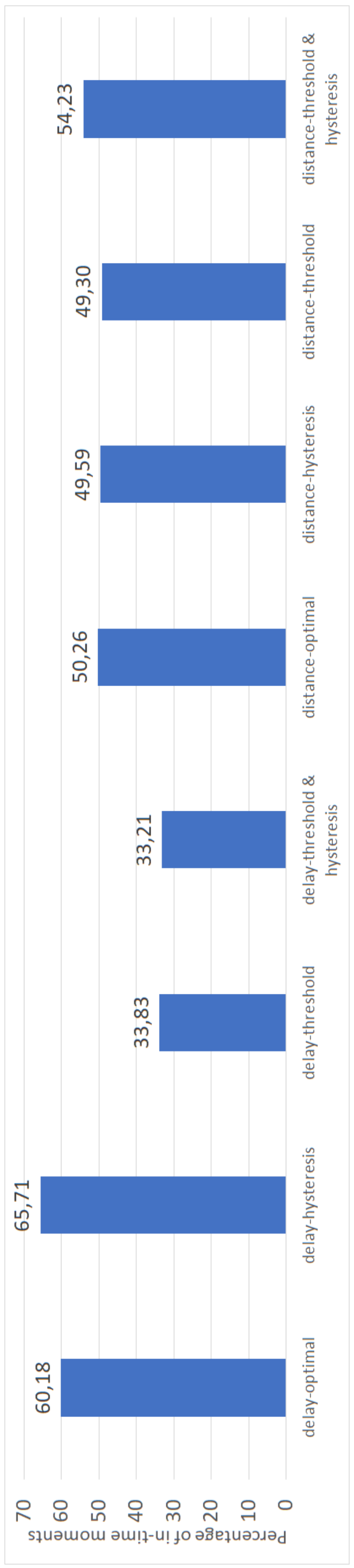

Figure 6.20: *Percentage of moments in which arrives in time.*

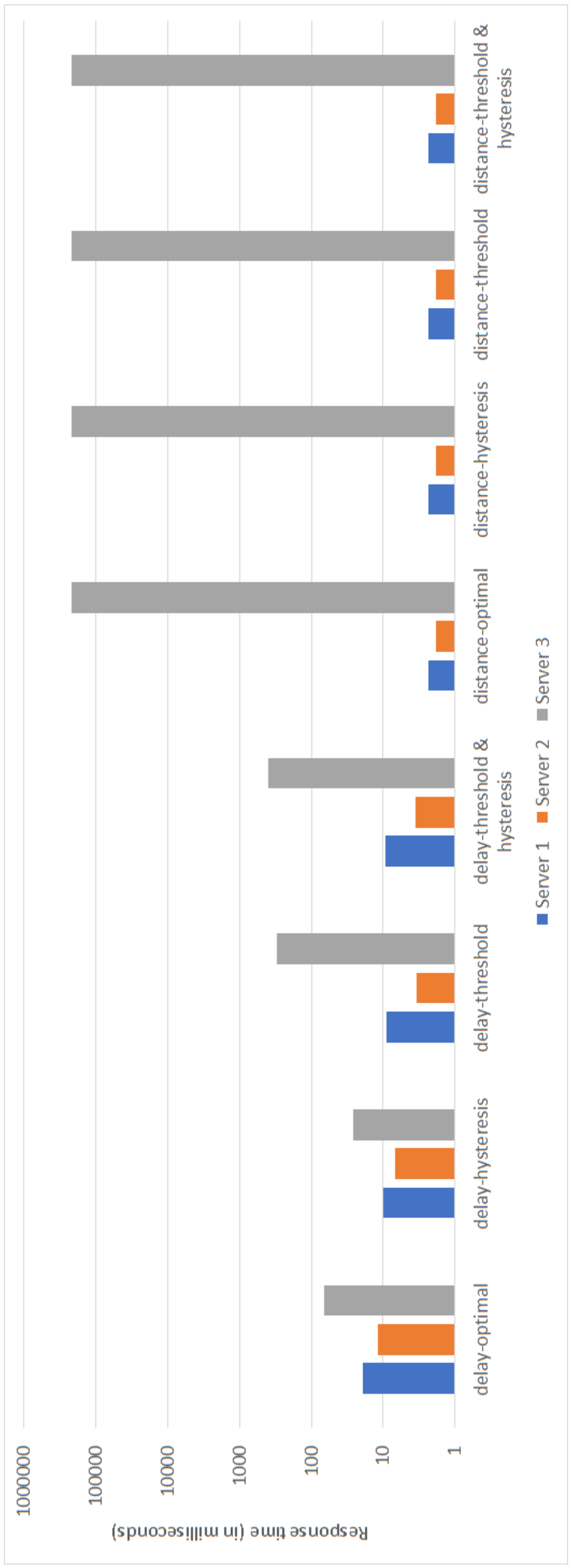

Figure 6.21: *Average server response time per strategy.*

Chapter 7

Conclusion

This work has attempted to find a way to determine the optimal data handover strategy for an automotive application in a MEC environment. As we have seen in Chapter 2, a lot of work has already been done in many areas of MEC research, including MEC applications and supporting technologies, as well as various techniques for performing a data handover. However, before this project, there was no work describing a strategy for when to make the decision to perform a data handover.

In Chapter 3 we define our research question: *What is the optimal handover strategy in an automotive MEC use case?* In Chapter 4 we then propose the following definition of optimal; *a strategy that is least likely to violate the application's RTT requirement.* We also pose that the best way to investigate this for this project with limited time is in a simulated environment using the tools ns-3 and SUMO. The use case for this research is TNO's platooning application, although the simulator is capable of mimicking many different applications. There are 8 strategies to be tested, a combination of two metrics (delay and distance) and four triggers (optimal, hysteresis, threshold, threshold & hysteresis).

Chapter 5 concerns the implementation of the simulation system. It describes the roles of UEs, MEC servers, and the orchestrator. It also describes in detail the processes that make up the simulator: requesting service, status reporting and data handover.

Our experiment results, discussed in Chapter 6, show us that physical distance to the server is not an optimal metric for data handover in a situation where individual MEC servers are likely to get overloaded. However, the results indicate that in a more homogeneous and powerful system, it could be a good alternative. Further experiments are needed to confirm this hypothesis. Nonetheless, such powerful servers require much more expensive hardware. We therefore believe that our current scenario is more likely to occur in the real world, outside of areas known to be very busy.

The distance-metric strategies show that it is crucial to system performance to perform a handover when a server starts getting overloaded. In fact, in our scenario, it is crucial to be aggressive in this respect; the two strategies with the highest handover frequencies, delay-optimal and delay-hysteresis, receive their replies on time nearly twice as often as the other delay-metric strategies. However, the downside to these strategies is that they have a more unstable RTT, though this effect is much more pronounced in the delay-optimal strategy than the delay-hysteresis. This instability can possibly be tampered by using a different set of parameters, particularly reducing the threshold; this must be confirmed with extra experiments.

There are two handover strategies with a significantly lower percentage of RTT violations than the others; the delay-optimal and the delay-hysteresis strategies. Of these two, the delay-hysteresis strategy has the highest portion of on-time moments at 65.71%. Furthermore, it is also the only strategy for which the average RTT is below the maximum acceptable 30ms, though barely. We therefore conclude that for TNO's platooning application, the optimal handover strategy is the delay-hysteresis strategy.

7.1 Contributions

This work has made several contributions to the research that is currently being done in this field. The primary contribution is that of a network simulator scenario for MEC. These sometimes exist, but are not always widely known, and do not always have multi-server capabilities. This work has provided a reasonably fast and simple way to test appropriate data handover strategies for a wide variety of applications. Finally, this work has given some insight into the factors of data handover that impact performance, e.g. the impact that a high handover frequency can have on various system parameters, as well as the impact that a very low handover frequency can have.

7.2 Future Work

This work provides a basic framework for MEC simulations. In future work, several features could be added or expanded to make it more realistic, for example:

- *Mobility.* The mobility models in this work are very bare-bones. The UEs drive in the same circle over and over. It would be interesting to see how the system performs if mobility is more complex, for example in an urban scenario with many possible routes.
- *Service duration.* In the current implementation, all jobs that arrive at the server have the same service rate, i.e. they take the same amount of time on average. It would be more realistic to have different rates for different types of jobs; a service request might require significantly more processing than a request for handover, or vice versa. This feature could also take into account message size, so that a service request message containing more data would take longer to be processed than a message carrying very little data.
- *Cross traffic.* In the current implementation, it is assumed that the application being tested is the only application running on the server. It is more or less possible to simulate that there are other applications running by diminishing the server capacity and asserting that the rest of the capacity is taken up by other applications. However, that means these other applications put a static reservation on the server capacity, which is not realistic. It would therefore be useful to implement dummy applications to be running on the same servers as the application under test is. It would be even more realistic if the number of these applications or the capacity they require could vary per server.

Finally, we recommend to run the experiments for different use cases. It is our hypothesis that what is optimal for our platooning application might not be optimal for e.g. a streaming application, but experiments are required to confirm or deny this.

List of abbreviations

Abbreviation	**Meaning**
ASP	application service providers
CPM	collaborative perception message
CSP	cloud service provider
DV	data volume
eNB	eNodeB
ETSI	European Telecommunication Standards Institute
FiWi	fiber-wireless
LTE	long term evolution
MEC	multi-access edge computing
MEO	mobile edge orchestrator
PGW	packet gateway
QoS	quality of service
RAM	random access memory
RSSI	received signal strength indicator
RTT	round trip time
SUMO	Simulation of Urban MObility
SWM	shared world model
TDMA	time-division multiple access
TNO	Nederlandse Organisatie voor toegepast natuurwetenschappelijk onderzoek (translation: Dutch Institute for Applied Scientific Research)
UE	user equipment
VM	virtual machine
VNF	virtual network function
5G-MOBIX	5G for cooperative automated mobility on X border corridors (European project)

Bibliography

[1] A. Ahmed and E. Ahmed. A survey on mobile edge computing. In *2016 10th International Conference on Intelligent Systems and Control (ISCO)*, pages 1–8, Jan 2016.

[2] Md Golam Rabiul Alam, Mohammad Mehedi Hassan, Md ZIa Uddin, Ahmad Almogren, and Giancarlo Fortino. Autonomic computation offloading in mobile edge for IoT applications. *Future Generation Computer Systems*, 90:149–157, 2019.

[3] Michael Till Beck, Martin Werner, Sebastian Feld, and Thomas Schimper. Mobile Edge Computing : A Taxonomy. In *AFIN 2014 : The Sixth International Conference on Advances in Future Internet*, 2014.

[4] Jakob Erdmann. Sumo's lane-changing model. In Michael Behrisch and Melanie Weber, editors, *Modeling Mobility with Open Data*, pages 105–123, Cham, 2015. Springer International Publishing.

[5] I. Farris, T. Taleb, H. Flinck, and A. Iera. Providing ultra-short latency to user-centric 5g applications at the mobile network edge. *Transactions on Emerging Telecommunications Technologies*, 29(4):e3169, 2018. e3169 ett.3169.

[6] Johanna Heinonen, Pekka Korja, Tapio Partti, Hannu Flinck, and Petteri Pöyhönen. Mobility management enhancements for 5G low latency services. *2016 IEEE International Conference on Communications Workshops, ICC 2016*, pages 68–73, 2016.

[7] Xueshi Hou, Yong Li, Min Chen, Di Wu, Depeng Jin, and Sheng Chen. Vehicular Fog Computing: A Viewpoint of Vehicles as the Infrastructures. *IEEE Transactions on Vehicular Technology*, 65(6):3860–3873, 2016.

[8] Stefan Krauß. *Microscopic modeling of traffic flow: Investigation of collision free vehicle dynamics.* PhD thesis, Deutsches Zentrum für Luft und Raumfahrt, 1998.

[9] Jani Lakkakorpi, Hannu Flinck, Johanna Heinonen, Pekka Korja, Tapio Partti, and Kalle Soranko. Minimizing delays in mobile networks: With dynamic gateway placement and active queue management. *IFIP Wireless Days*, 2016-April:1–3, 2016.

[10] Ang Li, Xiaowei Yang, Srikanth Kandula, and Ming Zhang. CloudCmp: Comparing Public Cloud Providers. *IMC '10 Proceedings of the 10th ACM SIGCOMM conference on Internet measurement*, pages 1–10, 2010.

[11] J Liu, J Wan, B Zeng, Q Wang, H Song, and M Qiu. A Scalable and Quick-Response Software Defined Vehicular Network Assisted by Mobile Edge Computing. *IEEE Communications Magazine*, 55(7):94–100, 2017.

[12] Pablo Alvarez Lopez, Michael Behrisch, Laura Bieker-Walz, Jakob Erdmann, Yun-Pang Flötteröd, Robert Hilbrich, Leonhard Lücken, Johannes Rummel, Peter Wagner, and Evamarie Wießner. Microscopic traffic simulation using sumo. In *The 21st IEEE International Conference on Intelligent Transportation Systems*. IEEE, 2018.

[13] Andrew Machen, Shiqiang Wang, Kin K. Leung, Bong Jun Ko, and Theodoros Salonidis. Migrating running applications

across mobile edge clouds. *Proceedings of the 22nd Annual International Conference on Mobile Computing and Networking - MobiCom '16*, pages 435–436, 2016.

[14] Francesco Malandrino, Carla Chiasserini, and Scott Kirkpatrick. The price of fog: A data-driven study on caching architectures in vehicular networks. In *Proceedings of the First International Workshop on Internet of Vehicles and Vehicles of Internet*, IoV-VoI 16, page 37âĂŞ42, New York, NY, USA, 2016. Association for Computing Machinery.

[15] ns-3, a discrete-event network simulator for internet systems. `https://www.nsnam.org/`. Accessed: 18-05-2020.

[16] T. Ojanperä, H. van den Berg, W. IJntema, R. de Souza Schwartz, and M. Djurica. Application synchronization among multiple mec servers in connected vehicle scenarios. In *2018 IEEE 88th Vehicular Technology Conference (VTC-Fall)*, pages 1–5, 2018.

[17] Milan Patel, Yunchao Hu, Patrice Hédé, Jerome Joubert, Chris Thornton, Brian Naughton, Ramos Roldan Julian, Caroline Chan, Valerie Young, Soo Jin Tan, and Daniel Lynch. Mobile Edge Computing âĂŞ Introductory Technical White Paper. *ETSI White Paper*, 11(1):1–36, 2014.

[18] J. Plachy, Z. Becvar, and E. C. Strinati. Dynamic resource allocation exploiting mobility prediction in mobile edge computing. In *2016 IEEE 27th Annual International Symposium on Personal, Indoor, and Mobile Radio Communications (PIMRC)*, pages 1–6, 2016.

[19] G. P. Pollini. Trends in handover design. *IEEE Communications Magazine*, 34(3):82–90, 1996.

[20] G Premsankar, M Di Francesco, and T Taleb. Edge Computing for the Internet of Things: A Case Study. *IEEE Internet of Things Journal*, PP(99):1, 2018.

[21] Mec for ns-3 code. `https://github.com/evavandeneijnden/mec_for_ns3`. Accessed: 28-11-2020.

[22] Bhaskar Prasad Rimal, Dung Pham Van, and Martin Maier. Mobile Edge Computing Empowered Fiber-Wireless Access Networks in the 5G Era. *IEEE Communications Magazine*, 55(2):192–200, 2017.

[23] Dimas Satria, Daihee Park, and Minho Jo. Recovery for overloaded mobile edge computing. *Future Generation Computer Systems*, 70:138–147, 2017.

[24] Aggeliki Sgora and Dimitrios Vergados. Handoff prioritization and decision schemes in wireless cellular networks: A survey. *IEEE Communications Surveys and Tutorials*, 11(4):57–77, 2009.

[25] Driving in circles. `https://sumo.dlr.de/docs/Tutorials/Driving_in_Circles.html`. Accessed: 04-09-2020.

[26] Xiang Sun, Nirwan Ansari, N. Edgeiot: Sun, X., & Ansari, Xiang Sun, and Nirwan Ansari. EdgeIoT: Mobile Edge Computing for the Internet of Things. *IEEE Communications Magazine*, 54(12):22–29, 2016.

[27] T. X. Tran, A. Hajisami, P. Pandey, and D. Pompili. Collaborative mobile edge computing in 5g networks: New paradigms, scenarios, and challenges. *IEEE Communications Magazine*, 55(4):54–61, 2017.

[28] Eva van den Eijnden. Research proposal: Optimal handover in multi-access edge computing for an automotive application. unpublished, 2018.

[29] S Wang, R Urgaonkar, K Chan, T He, M Zafer, and K K Leung. Dynamic service placement for mobile micro-clouds with predicted future costs. *2015 IEEE International Conference on Communications (ICC)*, 28(4):5504–5510, 2015.

[30] Shiqiang Wang, Rahul Urgaonkar, Murtaza Zafer, Ting He, Kevin Chan, and Kin K. Leung. Dynamic service migration in mobile edge-clouds. *Proceedings*

of 2015 14th IFIP Networking Conference, IFIP Networking 2015, 2015.

[31] Yu Xiao, Pieter Simoens, Padmanabhan Pillai, Kiryong Ha, and Mahadev Satyanarayanan. Lowering the barriers to large-scale mobile crowdsensing. *Proceedings of the 14th Workshop on Mobile Computing Systems and Applications - HotMobile '13*, page 1, 2013.