UNIVERSIDADE DE LISBOA

FACULDADE DE CIÊNCIAS

DEPARTAMENTO DE INFORMÁTICA

Design and implementation of a cloud-based membership system for vehicular cooperation

Tiago André Martinho Correia

Mestrado em Engenharia Informática

Especialização em Arquitetura, Sistemas e Redes de Computadores

Dissertação orientada por:

Prof. Doutor António Casimiro Ferreira da Costa

e co-orientada pelo Prof. Doutor Naercio David Pedro Magaia

2019

Acknowledgments

First, I would like to start by thanking my advisors, António Casimiro and Naercio Magaia. We worked closely together for the duration of the whole project, and this work would not be possible without them. I would like to thank João Pinto for similar reasons. Since we were building different parts of the same system, we had to coordinate a lot of things to make sure everything would be compatible for integration purposes. Things would have been a lot harder without your collaboration and help, thank you.

Second, I would also like to thank everyone from the group Fculianos: Francisco Araújo, João Batista, João Becho, João Loureiro, Nuno Burnay, Nuno Rodrigues and Pedro Vieira. I would like to thank all of you not only for being with me and helping me during this project, but also for all these years we spent together and all the projects we worked on since I started my Bachelor's degree. My academic life would not be the same without you and I am not even sure if I would be able to get this far, thank you.

Third, I would like to thank everyone from my group of friends House of Miró: Ana Tam, Diana Correia, Gabriel Marques, Joana Bernardo, Lisa Castro, Mariana Ferreira, Rebeca Gomes, Rita Osório, Rosa Silva, Sana Reis, Vebjørn Elvanes and Wooby. All of you are by far the most important people of my life and I have to thank you a lot for everything. Thank you for always being with me, thank you for the support and always being there, not only for the good but also for the bad moments. I would never be able to come this far without you.

Fourth, a very special thanks to Kotori, Riko and Arisa. There are no words to express what I feel for you. Thank you for being the rays of sunshine that light my life.

I would also like to thank every artist that created/performed most of the music I listened to while working. Some of the most important names are: $\mu's$, Aqours, Poppin'Party, Roselia, Pastel*Palettes, raise a suilen, Starlight Kuku Gumi, ClariS, Haruna Luna, Babymetal, Perfume, WACK, Twice, Red Velvet, GFriend, Oh My Girl, Orange Caramel, Secret, Apink, SNSD, as well as all the artists from Eurobeats compilations. Music is a big part of my life and everything is a lot easier because of it, thank you.

Special thanks to John Egbert and Kaname Madoka. Homura did nothing wrong.

Lastly, I want to thank Sofia Santos. I know we just met recently, but you are already someone extremely special to me. You helped me a lot during the last few days, and I can see you staying in my life for a really, really long time, thank you.

To all of you, thank you so much.

This work was supported by the Fundação para a Ciência e a Tecnologia (FCT) under LASIGE Strategic Project UID/CEC/00408/2019 and IRCoC project (PTDC/EEI-SRC/6970/2014).

“The little bird’s wings have finally grown, today is the day she takes flight”
Bokutachi wa Hitotsu no Hikari, $\mu's$

Resumo

Veículos pessoais tais como carros são o meio de transporte escolhido por grande parte da população, pelo que as nossas cidades hoje em dia estão construídas muito em volta deste meio de transporte. Tendo isto em conta, e a falta de investimento em outros tipos de transportes (tais como transportes públicos e ciclovias), é praticamente impossível viver sem automóveis em certas localidades, principalmente fora das grandes cidades e em países como Portugal.

Tendo em conta a quantidade de veículos que andam diariamente nas nossas cidades, os níveis de poluição e trânsito são maiores do que nunca e há filas intermináveis para ir a qualquer lugar, o que dificulta bastante a vida das pessoas em grandes cidades. Já foram sugeridas múltiplas propostas para resolver este problema tais como adicionar mais vias de trânsito, novos tipos de interseções mais eficientes, adicionar taxas de congestão, ou até mesmo banir certos veículos de certas localizações a certas alturas tal como já foi feito previamente na China. Infelizmente, nenhuma destas soluções funciona como esperado, e o problema acaba sempre por reaparecer e/ou até mesmo piorar. Gestão de trânsito é mais complicada do que parece, e é difícil arranjar uma solução que seja eficiente e funcione a longo termo.

Hoje em dia, os primeiros veículos autônomos estão a começar a aparecer, e dão-nos uma grande oportunidade para tentar resolver este problema. Os poucos veículos autônomos já existentes são simples e ainda não são uma opção viável para transporte diário, embora tudo mostre que isso esteja prestes a mudar em breve. Embora ajudem quanto à poluição e trânsito (já que não há o factor de erro humano e têm maior segurança), não resolvem o problema totalmente, pelo que esta solução só por si não é suficiente. Esses mesmos veículos autônomos, hoje em dia tomam as suas decisões com base apenas em sensores próprios e a perceção que têm do mundo exterior. Tendo isto em conta, estes veiculos não são perfeitos e há uma área não muito explorada que está em falta nos já existentes, a comunicação com outros veículos e/ou sistemas externos.

A comunicação entre veículos é um fator fundamental em falta que tem de ser explorado e considerado para a próxima geração de veículos autônomos. Se estes veículos tiverem a possibilidade de comunicar entre si, é possível que estes cooperem uns com os outros e que troquem informações úteis entre si, seja sobre o ambiente ou sobre tomadas de decisões.

Uma solução destas iria ajudar consideravelmente, sendo uma solução que teoricamente funcionaria mesmo a longo termo e reduziria bastante a poluição (já que os veículos circulariam com maior eficiência), o trânsito das nossas cidades, e a segurança dos passageiros, tornando as nossas vidas mais simples.

A este tipo de veículos é dado um novo nome: veículos cooperativos. Tal como o nome indica, veículos cooperativos são um subconjunto de veículos autônomos que não são totalmente independentes e dependem de infraestrutura externa e/ou comunicação com outras entidades de modo a executarem a sua condução autónoma e tomadas de decisões.

Existem duas abordagens principais para cooperação: os veículos comunicarem com outros veículos, ou usarem infraestrutura externa como intermediário para a comunicação.

Ambas estas abordagens têm várias vantagens e desvantagens. No caso de comunicação direta temos intervalos de comunicação muito baixos, o que é o ideal para um sistema destes, mas é necessario usar comunicações de baixo alcance, o que significa que há muitas falhas de comunicação já que os veículos estão constantemente em movimento. Outro problema é a falta de visibilidades, já que devido à natureza das comunicações de curto alcance, cada veículo só sabe informações de outros veículos bastante próximos.

No caso de usar infraestrutura como intermediário, a principal vantagem é o aumento da visibilidade, já que a infraestrutura pode ter informação de qualquer veículo, seja qual for a sua posição, mas não é adequado para situações em que é preciso comunicação rápida, como por exemplo manobras devido ao grande intervalo entre comunicações.

Tendo isto em conta, achamos que a melhor solução é uma mistura entre estas duas abordagens, tentando manter as vantagens de cada uma mas sem as suas desvantagens. Um sistema que segue essa abordagem mista já foi previamente proposto, mas ainda precisa de uma implementação, que é o que esta dissertação pretende tratar.

Este projeto apresenta uma possível solução para cooperação entre veículos autônomos usando como apoio um serviço na cloud. Os veículos comunicam a sua posição periodicamente a um sistema de membership na cloud que guarda e analisa esta informação de modo a ter uma visão global de todos os veículos nas estradas. Sempre que cada um destes veículos quer executar uma manobra, este envia um pedido para o serviço na cloud, de modo a receber a informação sobre os veículos em alcance de comunicação que são importantes para a realização da manobra em questão. Tendo conhecimento desta informação, cada agente tenta entrar em contacto com esses mesmos veículos usando comunicações sem fios de baixo alcance de modo a tentarem combinar a melhor e mais eficiente forma para a execução das manobras.

Tendo isto em conta, esta dissertação apenas tem como objetivos implementar a aplicação servidor usando um algoritmo de Membership muito básico, bem como desenhar e implementar o protocolo de comunicação cliente-servidor que vai ser usado para os veículos comunicaram com a aplicação na cloud.

A membership de um veículo é a lista de todos os veículos em alcance de comunicação que são relevantes para a execução de uma manobra.

Questões de escalabilidade foram pensadas durante a implementação, pelo que foi construída uma forma de poder dividir por vários servidores através de um conceito que criamos, os segmentos. Um segmento é uma zona bastante especifica e delimitada de um mapa 2D, cobrindo-o na sua totalidade. Todos os segmentos têm de ser o mesmo tamanho, e ter uma zona sobreposta com todos os segmentos adjacentes. Cada servidor de membership pode controlar um ou mais segmentos.

O protocolo de comunicação foi desenvolvido com uma aplicação cliente e uma aplicação servidor usando o Zookeeper, que é um serviço de coordenação usado para ajudar a criação e uso de ferramentas distribuídas. O Zookeeper é um servidor extra que serve de intermediário entre a aplicação cliente e a aplicação servidor. Periodicamente o cliente escreve no Zookeeper informações relevantes sobre o seu estado atual, tais como as suas coordenadas e velocidade. Periodicamente, o servidor vai buscar a informação sobre os veículos que controla ao Zookeeper, calcula a membership de cada um e coloca-as no Zookeeper. Quando cada veículo quer efetuar uma manobra, vai buscar a sua membership ao Zookeeper de modo a saber com que veículos tem de comunicar localmente para efetuar a manobra.

A implementação foi feita em Java, e foi feita da forma mais modular possível, de modo a poder facilmente adicionar novas funcionalidades, bem como integrar este trabalho com um algoritmo de coordenação externo.

Para efeitos de testes e demonstração, foram feitos dois tipos de avaliação, isto é/nomeadamente, avaliação funcional e avaliação de desempenho.

Para a avaliação funcional foi desenvolvida uma simulação para visualizar a aplicação a funcionar. Para usar a simulação, foi necessario integrar o trabalho desenvolvido com um protocolo de cooperação que usa a informação do sistema de membership para controlar os veículos presentes na simulação.

Para a avaliação de desempenho foram feitos testes de carga com a intenção de perceber qual é o bottleneck do sistema, e testes de execução, em que foi testado quanto tempo é que o servidor demora a calcular a membership para um numero incremental de clientes.

Palavras-chave: Veículo, Autônomo, Comunicação, Cooperação, Cloud

Abstract

Personal vehicles such as cars are the transportation method chosen by most people, and thanks to this, our cities are built around them, with roads that go to any place you could ever need to go.

Given the number of daily vehicles in our cities, the pollution levels and traffic congestion are higher than ever. Traffic makes everyone's life harder, and just creates more pollution, which ends up making living in a city a lot harder than it should. Multiples solutions have been proposed to help fixing this problem, but none of them work as expected or in the long run.

Nowadays, the first autonomous vehicles are starting to appear, and consequently, bringing the opportunity to once again, try to solve this problem. Current autonomous vehicles are simple and still not a viable option for daily transportation, but everything shows that is likely to change soon. They already help a lot with traffic and pollution, but sadly, not as much as we would like, which means it will not be enough in the long run and another solution is needed. The existing ones make their decisions solely based on their own sensors and nothing else. That is, it is the only view they have of the external world. Even considering this, these vehicles are still not perfect as there is still a subject that was not well explored, communication between vehicles.

Vehicle coordination is the next big step and an essential missing factor that has to be considered for the next generation of autonomous vehicles. By being able to communicate with each other, vehicles will be able to cooperate and share useful information about their own decisions or the outside environment.

A solution such as this would help considerably with our current traffic issue and we believe that this could be a long term solution with the advantage of reducing pollution (due to higher efficiency), higher passenger security, and making everyone's lives easier.

Keywords: Autonomous, Vehicle, Communication, Cooperation, Cloud

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Pollution and traffic congestion are two big problems that affect our daily life in cities and it is something that needs to be addressed as fast as possible. Multiple solutions to solve this have been suggested before, such as adding more lanes, new types of more efficient intersections, e.g., the Diverging Diamond Interchange [6], adding congestion charges, or even going as far as banning certain vehicles from certain locations at certain hours and/or days based on a very specific set of rules, e.g., it has been done previously in China [10].

All of these solutions help in the short term, but they will always end up coming back due to more traffic coming in as soon as people start seeing free space on the roads. This is something called Induced Demand [20], and it basically means that trying to solve traffic just creates more traffic, which means we need a different approach that works as a long term solution and ends up eliminating traffic congestion for good.

Vehicles are also becoming more sophisticated and nowadays they have a wide range of sensors and other tools available that allow them to do things that would be unimaginable a few years ago. Some examples are constant location tracking thanks to GPS, and detection of other vehicles and objects thanks to distance sensors.

Due to the latter, and more than ever before, autonomous vehicles are going to become everyday objects in a near future and a big part of our lives, influencing the way we interact with the world. Everyone talks about them, every big automotive company is working on them, they are important, and they are clearly the next big step in the transportation industry.

1.1 Motivation

Currently, autonomous vehicles rely mainly on data provided by their sensors, and that is the only way they have to get a view of the external environment. What is missing is exactly a way of vehicles communicating with each other, so they can cooperate while doing maneuvers. On the other hand, enabling cooperation is not an easy task as there are multiple factors to be taken into consideration. The biggest issue when considering

vehicular cooperation is how fast they are and how big the road system is. It is hard to cooperate when vehicles do not know who they are going to encounter and when. By using an external cloud service we could have a global view of the external environment, nearby vehicles, possible risks and threats, and with this, we could make predictions to aid vehicles on the road know what to expect and performing maneuvers.

Mobile cellular networks are also becoming cheaper, to the point where it is starting to be viable to have every single vehicle always connected to the internet, which is needed to allow the use of a centralized cloud service.

A system like this was previously proposed [5], and is currently just missing an implementation, which is what this project intends to do.

1.2 Objectives

The main goal of this MSc Dissertation is to design and implement a server application and a client-server communication protocol together with a simple Membership algorithm to be used in a Vehicle to Infrastructure (V2I) environment that can be used to aid vehicular maneuvers. This application needs to be scalable, so it can be used to manage thousands of vehicles, and modular so it can be easily extended or used by external systems.

In order to achieve this goal, we will break down the main objective into two separate objectives:

- To implement the membership service proposed in [5], designing the client-server protocol to use for the V2I communication and integrating it into the full project that includes the Vehicle to Vehicle (V2V) communication.
- To build a simulation scenario consisting of multiple vehicles and test cases to provide visual feedback and use it for evaluation and testing of the complete system.

1.3 Document structure

The remainder of this document will be structured as follows:

- In **Chapter 2** we present the state of the art in cloud computing, vehicular networks and other related concepts, together with some advantages and disadvantages of the existing and proposed solutions.
- In **Chapter 3** we present our approach to vehicular cooperation and propose a solution design, analyzing the problem definition as well as some details things to have in consideration.

- In **Chapter 4** we present our implementation of the proposed solution design, as well as a detailed description of its inner workings.
- In **Chapter 5** we evaluate the performance and viability of the implemented solution by doing performance tests as well as testing the system using a simulation.
- In **Chapter 6** we present our conclusions of the project as well as possible ways to extend it with future work.

Chapter 2

Context and related work

In this chapter we will introduce the state of the art on the subjects related to this project such as cloud services, membership services, autonomous and cooperative vehicles, and communication technologies.

2.1 Cloud services

According to National Institute of Standards and Technology (NIST) [21], cloud service is the name given to a service running on a remote machine somewhere in the world. The exact location is unknown (unless it is disclosed by the provider). This computing power is provided and can be changed on-demand, and is pooled among the different clients of the cloud provider.

Cloud Computing (CC) offers a lot of advantages, such as not having to worry about server maintenance or backups, hardware issues, replication, scalability as the cloud provider takes care of all of these issues. Given the fact that the location and internal details do not matter, users can focus on their main goal instead of having to worry about system details and maintenance.

There are three main CC models defined by NIST [21]:

Infrastructure as a service (IaaS) In this model, the cloud provider offers pay-as-you-go access to computing resources. The user is free to manage these resources at will (ie, Virtual Machines (VMs))

Platform as a service (PaaS) In this model, the cloud provider offers an online platform where the user can deploy and run their own apps (i.e., JVM, .NET Framework, etc). The user does not have direct access to the computing resources or machine settings, but has full control over the deploying environment.

Software as a service (SaaS) In this model, the cloud provider offers access to only a very specific piece of Software that the user can use in a remote environment (i.e.,

Feature \ Model	Self-hosted Software	Infrastructure as a Service	Platform as a Service	Software as a Service
Applications	User	User	User	Provider
Data	User	User	User	Provider
Runtime	User	User	Provider	Provider
Middleware	User	User	Provider	Provider
OS	User	User	Provider	Provider
Virtualization	User	Provider	Provider	Provider
Servers	User	Provider	Provider	Provider
Storage	User	Provider	Provider	Provider
Networking	User	Provider	Provider	Provider

Table 2.1: Cloud models management differences

Google Docs, email). The user does not have control over anything other than what the Software provides

Table 2.1 compares what is managed by the user and the Cloud Provider for each of these models as well as usual self-hosted software.

2.2 Membership services

Group Membership systems detect entities in a group and keeps their information updated. These kind of systems keeps track of existing members of a group, as well as the ones entering or exiting, either by quitting or crash.

This information is important in any kind of system that needs constantly updated information of all the available members inside a group, which means that an efficient group membership protocol is needed.

Knowing accurate group membership is especially complex when the system is highly dynamic and entities are constantly entering and leaving. Group membership for vehicular networks is one of these cases, which gets even more complex due to another issue: V2V communications will often fail, which means that in most cases, contacting other vehicles is hard, except in very specific situations such as when all the relevant vehicles are driving in the same direction at about the same speed, or are stopped. Given this, relying only on V2V communications to keep track of group membership is not reliable and another solution should be considered. Contacting a cloud system and using it to save the membership information should be a great solution here since cellular communications are more reliable and should be available most of the time. A cloud system providing vehicles the correct membership already accounting delays could also give them the opportunity to actually use V2V communications reliably by improving considerably the amount of failed connections.

There are several proposals and implementations of efficient group membership protocols such as SWIM [8] and JGroups [15] which are libraries used for reliable one-to-one or one-to-many communication by providing tools that allow sending messages to groups of processes and automatic detection of new, removed and crashed members.

2.3 Autonomous driving

Autonomous vehicles are a special category of vehicles that can drive with little or even no human interaction or input. This kind of vehicles use a wide variety of sensors to collect information and perceive the environment around them. This can range from computer vision, radar, sonar, inertial measurement units to GPS and odometry systems. All the data collected by these sensors is collected and processed together by computer algorithms to identify road signalization, possible obstacles, other vehicles and navigation paths.

This category of vehicles raises a lot of questions, mostly ethical and in terms of security. In case of a crash, who is responsible, the driver or the vehicle manufacturer? Are autonomous vehicles safer than the ones controlled by humans? It can be said that there is a disadvantage in terms of security, however autonomous vehicles will always end up being more safe since there is little human interaction/input, which removes human error. There are thousands of road accidents each year, and this can easily be reduced considerably or even avoided with autonomous vehicles.

The Society of Automotive Engineers (SAE International) defines five levels of automation [18] as seen in table 2.2. As of 2018, we can say we are at level 2 [25], where the most advanced commercial vehicles have an auto pilot mode that allows it to control itself and take some decisions under certain circumstances while still needing a human driver as a fallback to take most decisions and making sure everything works as expected. As automation technology evolves, we are slowly approaching level 3 and we can see technologies such as these starting to appear on new vehicles like the Tesla Model 3 [27].

2.4 Cooperative vehicles

According to SAE International [18], Cooperative Vehicles are a specific subset of Autonomous Vehicles that are not self sufficient and depend on outside infrastructure and/or communications with outside entities to perform their autonomous driving, even if they fallback to their sensors in case of communication errors.

The main advantage Cooperative Vehicles offer in favor of Autonomous Vehicles is that by sharing decisions and knowing what others will do, we can considerably reduce waiting times and increase the overall speed of vehicles, which would drastically improve the traffic flow as well as save fuel and reduce pollution.

Level	Name	Narrative Definition	Execution of steering and acceleration/ deceleration	Monitoring of driving environment	Fallback performance of dynamic driving task	System capability (driving modes)
Human driver monitors the driving environment						
0	**No Automation**	The full-time performance by the human driver of all aspects of the dynamic driving task, even when enhanced by warning or intervention systems"	Human driver	Human driver	Human driver	n/a
1	**Driver Assistance**	The driving mode-specific execution by a driver assistance system of either steering or acceleration/deceleration" using information about the driving environment and with the expectation that the human driver performs all remaining aspects of the dynamic driving task	Human driver and system	Human driver	Human driver	Some driving Models
2	**Partial Automation**	The driving mode-specific execution by one or more driver assistance systems of both steering and acceleration/deceleration using information about the driving environment and with the expectation that the human driver performs all remaining aspects of the dynamic driving task	**System**	Human driver	Human driver	Some driving Models
Automated driving system ("system") monitors the driving environment						
3	**Conditional Automation**	The driving mode-specific performance by an automated driving system of all aspects of the dynamic driving task with the expectation that the human driver will respond appropriately to a request to intervene	System	**System**	Human driver	Some driving Models
4	**High Automation**	The driving mode-specific performance by an automated driving system of all aspects of the dynamic driving task even if a human driver does not respond appropriately to a request to intervene	System	System	**System**	Some driving Models
5	**Full Automation**	The driving mode-specific performance by an automated driving system of all aspects of the dynamic driving task under all roadway and environmental conditions that can be managed by a human driver	System	System	System	**All driving modes**

Table 2.2: SAE Automation Levels. Taken from [18]

Technically, if a cooperative vehicle is able to fallback to only sensory data and perform correctly in case of communication errors, there are no disadvantages other than adding more complexity to the system.

Cooperative vehicles also raise a lot of security questions. Can we really trust information given by other vehicles? What if a malicious user purposely sends wrong information or compromises existing vehicles? These are big issues that need to be solved before we can consider this model.

2.5 Technologies & maneuvers

Maneuvers are one of the biggest sources of road traffic [2]. Every time a vehicle needs to perform a maneuver, the driver has to slow down and/or stop, pay attention to the road and other elements around it, and finally perform the maneuver if all the conditions are met. On top of that, due to all these necessary steps and the inevitability of human error, maneuvers are also a big contributing factor for most road accidents [12].

One of the main goals of autonomous driving is exactly helping with maneuvers, trying to make them more efficient while also making them faster and safer. We will now briefly list some of these maneuvers:

Platooning Platooning [4] consists of having a platoon of vehicles that are following each other in a straight line, with all of them going at the same speed, and with a given

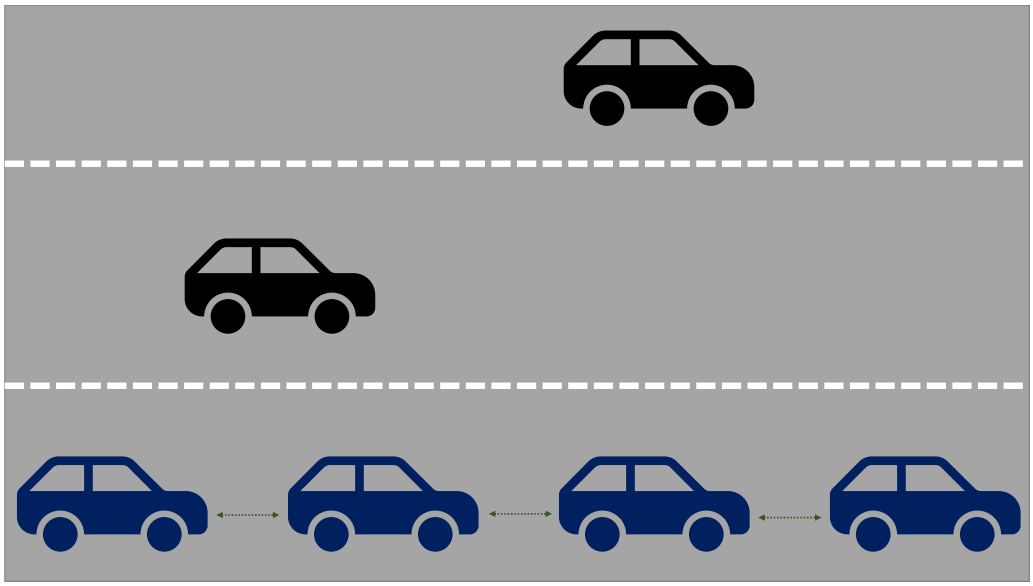

Figure 2.1: Example of a platoon

safety distance from each other. The front vehicle controls the platoon and defines variables such as speed. The main idea is being able to avoid traffic congestion since the platoon is a controlled environment where every vehicle that is part of it can drive at high speed without having to worry about interactions with other vehicles. This helps reaching higher speeds while spending less fuel due to less air resistance. There are lots of other advantages and disadvantages of platooning, but that is out of the scope of this work. Figure 2.1 shows an example of a platoon.

Lane Changing Lane changing can be subdivided into a lot of other different maneuvers such as going in or out of a platoon, getting out of the main road by changing lanes, or getting into the main road. In the case of platooning, vehicles who want to get in notify already participating vehicles who will open a spot in the middle of the platoon for them. It is also possible to join at the end. To leave the platoon, a vehicle notifies the other members of the platoon, changes lane, and the vehicle right after it closes the distance.

Intersection Crossing Intersections are one of the biggest source of accidents and traffic. Therefore finding a good solution to help with these maneuvers is a big priority. By having knowledge of nearby and unsafe vehicles and being able to communicate with them, most of these accidents can be avoided. We can decide who crosses the intersection and when they do it. With a system like this, theoretically there would be no need to stop at intersections, that is, we could keep the traffic flowing and avoid unnecessary congestion.

Roundabout The main idea of roundabouts is to facilitate intersection crossing, how-

ever, they also come with their own set of issues. Crossing a roundabout involves stopping at the entrance and waiting to get in, followed by multiple maneuvers of lane changing. All of this slows down traffic a lot on top of creating multiple opportunities for accidents. Autonomous vehicles can make a huge improvement here, by reducing waiting times and making lane changes more fluid, efficient and safer. Figure 2.3 shows an example of a roundabout problem.

Virtual Traffic Lights Virtual traffic lights are not necessarily maneuvers, but are a technology that can considerably help most maneuvers previously mentioned. Systems like the one proposed in [11] suggest the removal of physical traffic lights in favor of virtual ones that use V2V communications to control the traffic flow. Systems such as these offer a lot of advantages since they can be more dynamic than physical traffic lights, adapting in real time to traffic, reducing waiting times, and increasing the flow of traffic.

Virtual traffic lights offer a great opportunity for removing physical infrastructure of our roads while also improving traffic flow considerably. This could also be supported by the cloud, which could improve even further the efficiency of this system. Figure 2.2 shows an example of an intersection problem and how a system like virtual traffic lights could help with traffic flow.

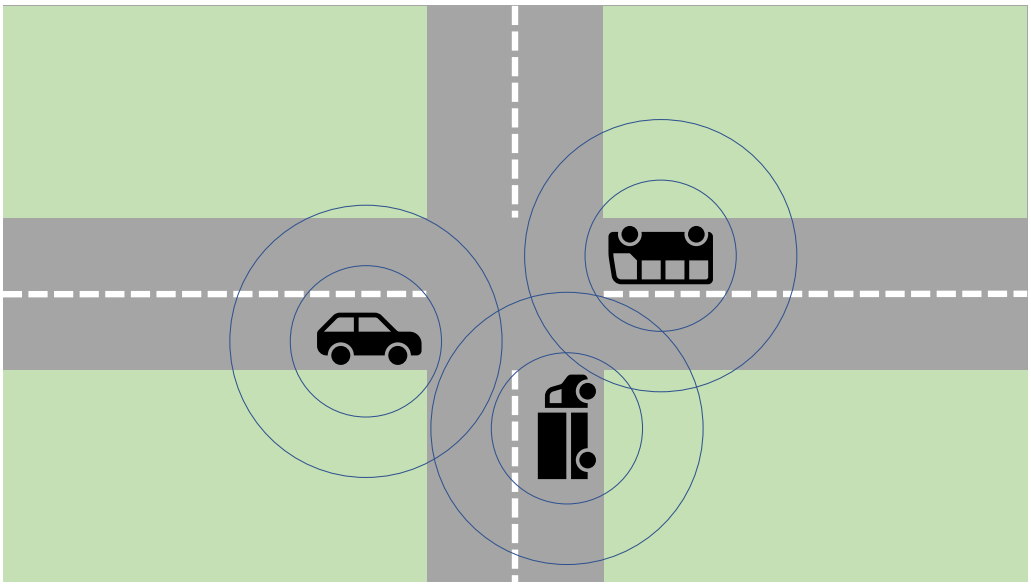

Figure 2.2: Example of an intersection problem and how a system like Virtual Traffic Lights could help

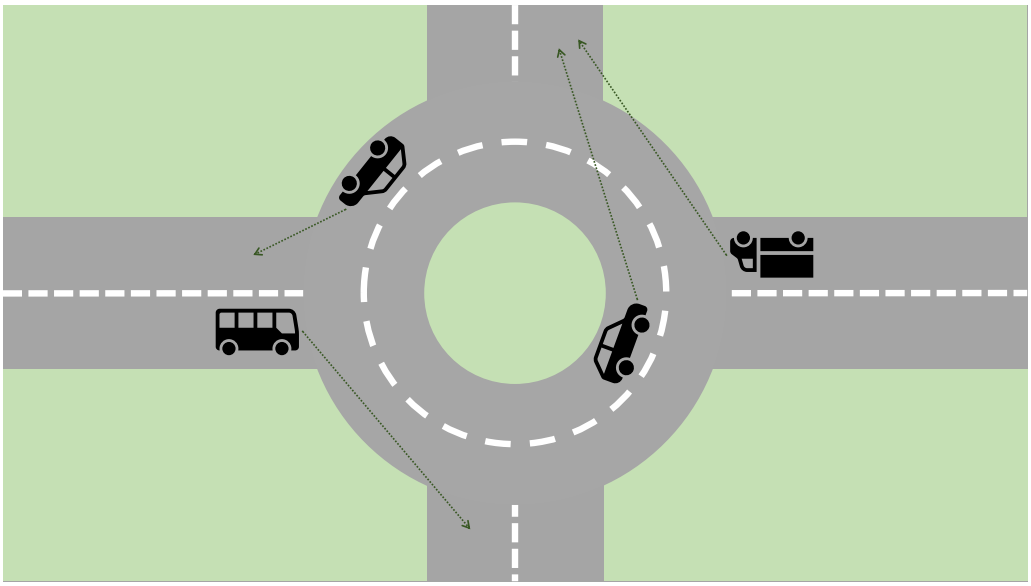

Figure 2.3: Example of efficiency issues with roundabouts

2.6 Communication

For vehicular communication, the main available option is working with wireless technologies, since connecting wires to high speed moving vehicles is not a viable solution.

Wireless communication technologies are obviously not as reliable as wired ones, so we have to choose well which one we are going to use. In terms of communication, we need to think about two types of approaches: V2V and V2I.

The main option for V2V is using short-range communications considering the fact that these provide high bandwidth with low latency, which is a requisite for communication between two moving vehicles.

For V2I, there are three main options: terrestrial communications, satellite communications or using road stations while communicating with them using short-range communications, the same technology used to communicate with other vehicles.

2.6.1 Short-range communications

There are multiple protocols and standards for short range communications, but the most relevant one for this work is IEEE 802.11p [17], which was specifically made with vehicles in mind. This protocol uses frequencies in the 5.9 GHz, with transmission speeds up to 20 Mbps and a transmission range of up to 1 km while withstand vehicle speeds up to 260 km/h [19].

Mobile ad hoc networks (MANETs) [7] are an important concept in terms of short-range communications. This is a type of network composed by mobile devices (nodes) that are connected to each other wirelessly in short-range. Each node constantly connects to all the other nodes in range and uses them as intermediates to deliver messages to other

nodes that are part of the network. This type of network does not need infrastructure and is constantly changing given that the nodes are always moving. Vehicular ad hoc networks (VANETs) [30] are an extension of MANETs where nodes are vehicles.

2.6.2 Terrestrial communications

In terms of terrestrial cellular communications, we currently use fourth generation technologies such as LTE which enables theoretical peak speeds up to 1 Gb/s [13]. This is good for our current usage, but starts being an issue as Internet of Things (IoT) devices become more popular due to higher bandwidth demand to the public infrastructure. This is also an issue on really dense populated areas or events where everyone has at least one connected device. A good example of this are big sport events that often need to be supported with extra temporary antennas to support the influx of people while keeping the network stable.

This is obviously a big issue, and it means that we still can not fully depend on LTE for our vehicles, as adding it to every vehicle could easily break the network similarly to how these big events do.

Currently, the first 5G enabled devices and infrastructure are starting to show up, and will be a big improvement to the current 4G infrastructure due to much higher bandwidths and latencies [3]. We can see this in specifications like IMT-2020, which promises peak speeds up to 20 Gbit/s and latencies of 1 ms [22]. Vehicles cellular communication could benefit tremendously from this as 5G will allow a lot more simultaneous connected devices which means having every single vehicle connected to the internet will finally be viable.

2.6.3 Satellite communications

Internet satellite communication options exist, but they are not the best since current solutions suffer from big delays while being more expensive and still having some coverage issues [26]. Most of these options are in geostationary orbit, which means that communicating with them really presents big delays due to the distances. This also means big dishes are needed and that the communications are seriously affected by weather conditions, i.e., they are not reliable for constant communication or to use on ”small” moving objects such as vehicles.

Furthermore, there are proposals like SpaceX’s Starlink constellation system that proposes a constellation of 12 000 low Earth orbit communication satellites that would offer low latency global internet with high coverage and speeds at a lower cost than other satellite options [14]. This system will use smaller phased array antennas for up and downlinks and laser communication between satellites to provide global low-latency high bandwidth coverage. A system like this is perfect for vehicles since it guarantees that they have

global coverage with low latency, no matter the location. Starlink already started tests and is currently scheduled to be operational somewhere around 2023 [9].

2.6.4 Cloud communications

Previous literature defines three big types of cloud networks for vehicular operation. There are good sources [29] with brief descriptions of these approaches:

Vehicular Cloud is a local cloud established among a group of cooperative vehicles. An inter-vehicle network (i.e., a VANET) is formed by V2V communications. The vehicles in a group are viewed as mobile cloud sites and cooperatively create a vehicular cloud.

Roadside Cloud is a local cloud established among a set of adjacent roadside units. In a roadside cloud, there are dedicated local cloud servers attached to roadside units (RSUs). A vehicle accesses a roadside cloud by Vehicle to Road (V2R) communications.

Central Cloud is a cloud established among a group of dedicated servers in the Internet. A vehicle accesses a central cloud by V2R or cellular communications.

As seen in [28] and [23], most of the proposed solutions are usually based on either VANETs or vehicular clouds, and there are not many that use a central cloud. This was not an option up until now, as having an internet connection in vehicles was not an economically viable option until the last few years. There are not many solutions that use always connected vehicles that use a central cloud.

THe same article [23] goes even further and suggests a possible use for Vehicular Clouds could be using the wasted computing resources of vehicles as a cloud. This concept is interesting as it could allow the computing power of a real cloud infrastructure, but physically closer to where it is needed, which means less communication delays.

However, there are a few problems with some of these current solutions based on VANETs or vehicular clouds:

- Mainly, the issues and disadvantages that the Peer to Peer (P2P) model has against the client-server model.
- Not having a central cloud, drastically limits how much each vehicle can see. Without it, there is no such thing as a global view of the traffic.
- If we use a vehicular cloud, the data we have depends a lot on our environment and the amount of vehicles around us, which is something that is always changing or can even be non existing. This is not reliable and it is never guaranteed that we will have all the needed data, or that it is updated.

- Any of these solutions will have more communication errors than the central cloud approach due to the short range communications used while moving at high speeds.
- If we use a Roadside Cloud, we will not always be connected to the cloud, which means that every time we pass by a checkpoint we need to transmit all the necessary information to that vehicle. This will never be as accurate as a persistent connection, and depending on the distance between checkpoints predicting possible interaction between vehicles will be considerably harder.

On top of not having these issues, there are also some advantages of having a central cloud:

- We know the full state of the network, which means we can easily predict conflicts with other vehicles and other risks, even if these vehicles are not in range of each other at the time of prediction.
- Since we have a persistent connection, this means we can query the central server very frequently for all the information we need.
- Guarantees that the state given by the central server is accurate.
- More efficient data transfer, since we can just get all the information needed from the server at once.
- A higher level of security since we mainly only need to trust on a single entity to get a fully updated state (assuming the server is not compromised).
- Higher reliability since it is guaranteed that the state we get from the server is fully updated (assuming the server is fault tolerant).

Another interesting idea mentioned in this article [23] is the concept of Network as a Service where the main goal is to offer internet to other vehicles on the move. In the use case mentioned, the shared network would be expected to just give internet access to other vehicles in range, but in our use case, this concept is extremely useful and we can take it a step further and use it to give system information to vehicles that for some reason do not have an internet connection. This could help either in the case of an error, system incompatibilities or legacy devices that are not part of this system in particular. This could help with maneuvers, even if the vehicle does not belong to our network and/or membership, which would help with the adoption of our system.

Chapter 3

Membership service design

In this chapter we present the design of our proposed solution. We start by presenting the problem definition, followed by the system assumptions. After that we describe the membership protocol, followed by more specific details, such as the Membership concept and how we achieve scalability.

3.1 Problem definition

The main idea of this dissertation is implementing a cloud-based membership system that was previously proposed in [5]. This system consists of a vehicular cooperation system aided by a cloud-based service that vehicles send information to. By having relevant information about each vehicle such as their position, speed and intentions, this cloud-based membership service must be able to provide useful data about nearby vehicles that are relevant for their maneuvers. Ensuring that this information is always updated is the problem we want to solve.

The next sections of this chapter will describe important concepts and information of this proposed system [5] to examine what needs to be done and implemented on the following chapter.

3.2 Assumptions

The following assumptions were considered:

- The clocks of every vehicle as well as the cloud system are synchronized via GPS.
- There are no malicious clients.
- The membership provided by the server is always accurate, as long as it is inside its validity period.
- The road is divided into segments with fixed sizes.

- Every vehicle inside a segment is in communication range of every other vehicle inside the same segment. This assumption is explained on 4.2.
- Every vehicle inside a segment is important for a maneuver happening inside that same segment.

3.3 The membership protocol

We call membership to the group of vehicles in communication range of a given vehicle that are important for the performance of a given maneuver.

The cloud service aims at keeping track of all the vehicles on the road and their current location. This information is sent periodically to the cloud by the vehicles, to make sure it is updated, which also query the cloud for data whenever they want to perform a maneuver. This data is called the Membership which is essentially a list of other vehicles in communication range of a given vehicle wanting to execute a maneuver. This list contains, for each vehicle, information about their current coordinates, velocity and acceleration. Other than this list, this membership structure also contains a flag called the maneuver opportunity, which the server sets as true if all the conditions are met for the performance of a maneuver. If the maneuver opportunity is true, and having this information, each vehicle can try to communicate directly with each of the other vehicles with the goal of cooperating to execute safe and efficient manuevers.

This dissertation focus on the cloud service portion of this system as well as the client-server communication, while the V2V communication part of the system is out of our scope.

The technology used for vehicles to communicate with the cloud service is not set, but will most likely be cellular based such as 4G or 5G. For more information about vehicular communication, please see section 2.6.

The communication protocol between each vehicle and the cloud service is also not defined and it needs to be designed before the implementation. We do this in section 4.2.

The membership server calculates the membership of an agent by using two functions:

getUnsafeAgents() For a given agent, returns a predictive analysis of possible conflicting situations with any other known agents within a given time horizon, in case the target agent wants to do a maneuvers.

getReachableAgents() For a given agent, returns a predictive analysis of all the agents that will be in communication range of that agent within a given time horizon.

At any point in time, for any given agent, its membership is a list of all the vehicles that besides being unsafe are also reachable.

In the current design, reachable agents are just all the agents inside the same segment, while unsafe agents are all the reachable agents.

Whenever the membership is calculated it is given the current timestamp, and a validity period is calculated using this initial timestamp and a know constant. Vehicles are only able to use the information provided by the membership server within this validity period. The membership server is supposed to keep a valid membership available all the time, but in case the vehicle does not get it, either because of a crash or a communication failure, and ends up without a valid membership it will fallback to normal autonomous behavior.

The server calculates the membership periodically for every single agent to make sure there is always a valid one available as soon as a client needs it.

3.4 Scalability

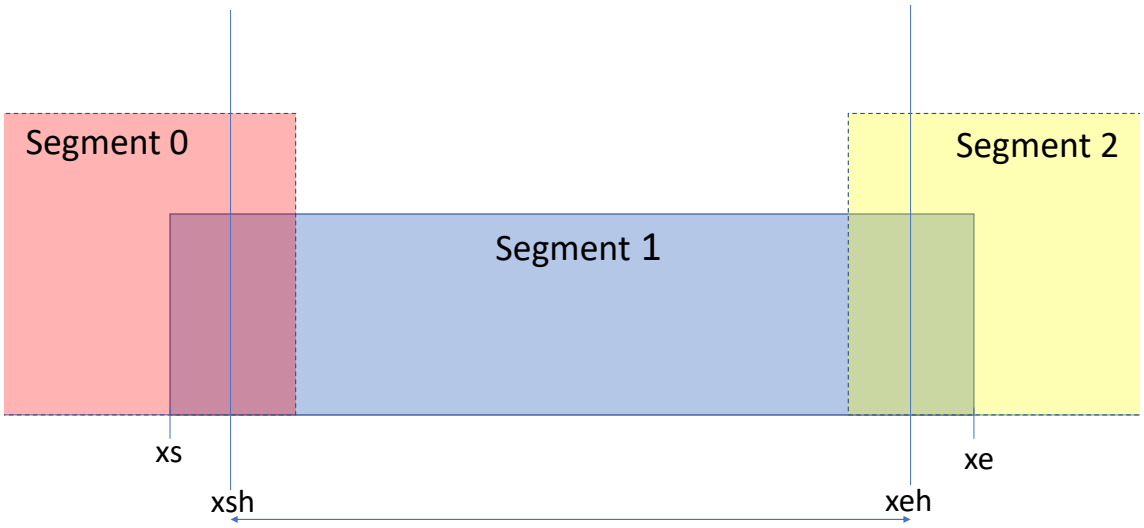

Figure 3.1: Road segments partitions

One of the main goals of this work is scalability, and making sure we would be able to spread the server application across multiple machines is a great way to achieve it.

To make this possible, we came up with the road segments concept that is illustrated in Figure 3.1. The main idea behind it is dividing the road map into multiple independent segments that cover the entire map. Each segment can have any number of adjacent segments, and there will always be an intersection zone with every single one of them where they overlap. The size of each segment is not defined and depends on the implementation, and the servers where membership service is deployed on.

Each server of the membership cluster is supposed to take care of a single segment, but they should also be able to controls more than one whenever necessary.

There is an important point in the middle of the intersection between two segments called the handover point (`xsh`).

Each server only generates membership data for vehicles between the two handover points of a given segment, that is, everything from `xsh` to `xeh`, but when calculating the membership for a given vehicle inside a segment, it takes in consideration the full segment, i.e., from `xs` to `xe`.

The main idea behind this is that each server should have full control over a precise zone, but that other vehicles close to it but outside its control zone should also be considered for predictions. This way, changing segments should be as transparent as possible as both servers should take the vehicle into consideration even though only one controls it.

Chapter 4

Membership service implementation

In this chapter we will walk through the implementation of the solution design proposed in the previous chapter. We start by presenting the implementation challenges, then we present the middleware choices, followed by a detailed view of the software architecture, concluding with details about the solution implementation.

4.1 Implementation challenges

There are multiple implementation challenges to take into consideration while building a service like this.

The first challenge is processing the information. There are lots of vehicles on the road, which means there is a lot of information to process. As the total number of vehicles increases, the time needed to generate the membership of each vehicle also increases. This is an exponential grow, which means performance is an important factor to take into consideration and the information processing needs to be done as efficient as possible.

Another important challenge is the choice of a middleware to use for communication between the vehicles and the membership service server.

The choice of programming language is also important as a language with good libraries and full compatibility with the chosen tools and middleware is desirable.

Another challenge has to do with the fact that the developed solution needs to be integrated into the full system that includes the coordination software used by the vehicles. After integrating it, a simulation needs to be built to evaluate the whole system. The Robot Operating System (ROS) is planned to be used as the middleware between the full application and the simulator.

These requirements have to be taken into consideration during all the implementation period and will heavily influence the programming language used and the decision to make the whole system as modular as possible.

4.2 Middleware

For the communication between the server and the client application there are two main possible approaches:

Direct communication from the clients to the server

Intermediate repository between the server and the clients that acts as a middleman and stores data that can be accessed asynchronously in any point in time by any of the parties.

Direct communication offers less delays, as there are is no middleman, reduces system complexity and removes a possible bottleneck compared to the intermediate repository approach. However, having an intermediate repository also offers its set of advantages over direct communication, such as separating the data processing from data management, which helps with availability as data operations are independent from how busy the server is. Having all the data in a centralized location also creates the option to easily have multiple servers, which is something needed to provide scalability.

Given these advantages and how important scalability is for this service, we ended up going with the intermediate repository approach, and decided to use the Zookeeper [16] as our middleman. The latter is a coordination service for distributed systems, and it provides a set of primitives these systems can use to implement complex tasks like synchronization, configuration, groups and naming. It also comes with built-in support for being deployed on a cluster, which means it is qualified to deal with possible availability and scalability issues.

It uses a hierarchial tree structure to organize data. Each tree note is a key/value pair where the key is a string and the value is binary, which means it can store any kind of information.

One of the most useful Zookeeper features is the concept of ephemeral nodes. These nodes are tied to the session of a client and are automatically erased when the client that created them closes the session or stops answering, being this due to a crash or some other reason. This is great for our system because as it provides an intuitive way of keeping track of all its active clients.

Given all of this, ZooKeeper is the perfect tool for our implementation, and Figure 4.1 shows how we organize the data using this structure.

The black node (/) is the root of the structure. The red and orange nodes are parent nodes and do not store any data.

Whenever a client pushes its information to Zookeeper, it creates/updates a node under agents where the key is its Universally Unique Identifier (UUID) and the value is a serialized object containing all the relevant data. Since these nodes are Ephemeral, only

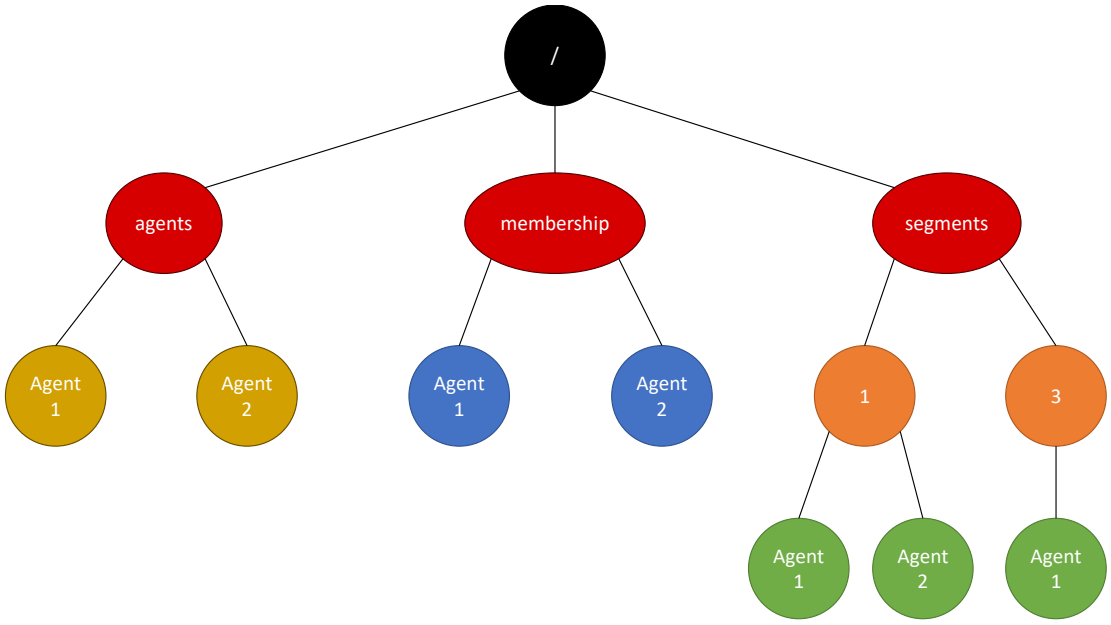

Figure 4.1: Zookeeper structure

active clients will be present. It also creates a node under `/segments/<id>` for every segment it is part of, where id is the id of the segment. These nodes are basically only used as a pointer to the main ones under `/agents/`. The nodes created under `/segments/<id>` are created with a Time to Live (TTL), which means they are automatically deleted when the TTL time ends. This time is defined by the client application and it depends on how frequently it pushes data to Zookeeper.

Whenever the server wants to fetch data, for every segment it controls, pulls the list of child's of `/segments/<id>`, where id is the segment id. With a list of agent id's available, the server starts pulling the data of each agent from `/agents/`.

With the data of every agent, the server can determine the membership of each one of them. After completing this task, for each agent, it creates a node under `/membership/` where the value is the agent id and the value is a serialized membership object. With this data available, the client application can pull it from Zookeeper whenever it needs to perform a maneuver.

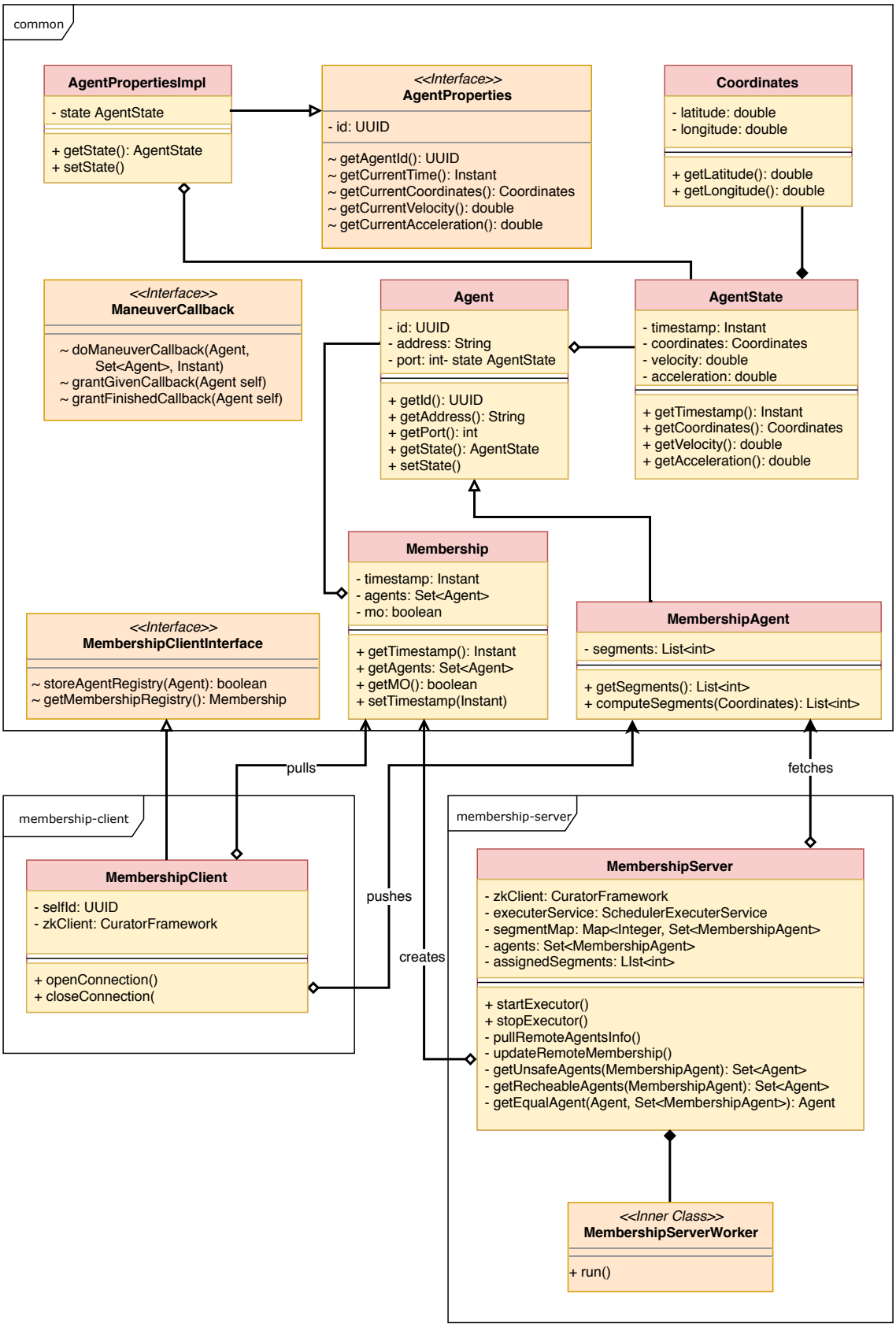

Figure 4.2: Class diagram of the membership client and server

4.3 Software architecture

Figure 4.2 shows the joint class diagram of the client and server applications. The client interface exposes two functions:

boolean storeAgentRegistry(Agent agent) Called whenever the client wants to update its information on the membership service. Internally, the implementation of this function pushes the Agent object to the Zookeeper server using the approach described on section 4.2.

Membership getMembershipRegistry() Called whenever the client needs to pull its membership information. Internally, the implementation of this function pulls the Membership object from the Zookeeper server using the approach described on section 4.2.

The `MembershipClient` object that implements this interface also provides two more functions `openConnection()` and a `closeConnection()` that, respectively, open and close the connection to the membership service, which provides extra control over when the system is active.

The server application has a `startExecutor()` and a `stopExecutor()` function, which,respectively, start and stops the connection to the Zookeeper server and the server executor. This executor executes at a fixed rate and periodically pulls segment and agent data from Zookeeper, processes it, and pushes the generated membership data to Zookeeper using the model explained on section 4.2. The timer used by the executor for the interval between each execution can be defined when starting the server application.

4.4 Language selection

Given the several requirements of the projects, choosing a programming language was not easy. In this section we start by presenting our possible options, followed by an explanation of the decisions that lead us the the final implementation.

4.4.1 Requirements and options

We had a few requirements to choose a programming language language:

- Allow the use of ROS
- Allow the use of Zookeeper
- Use the same language used for the Coordination Protocol

Given that ROS only has C++, Java and Python libraries, and that Zookeeper only has C and Java Libraries, the only viable choices would be C++ and Java. However, the Zookeeper C library does not have an official C++ version available, which means we would need to build a Wrapper around the C library.

At the time, the ROS Java library only has official support for ROS Kinetic, which officially only supports Ubuntu 16.04, which meant we would need to use an older version of both Ubuntu and ROS.

4.4.2 Selection process

Given that we did not want to use an old version of ROS, nor downgrade the operating system, initially we decided to use C++.

The first task of developing a C++ implementation was getting familiarized with the ROS building environment, which ended up taking some time due to the unfamiliarity with C++. This task ended up being completed successfully, and we ended with a functional ROS publish/subscribe module.

The second task was creating a wrapper for the C Zookeeper library. This task turned out to be a lot more difficult than expected. The C library is really low level and lacks some of the most advanced features and use cases of Zookeeper, which meant that this also had to be build on the wrapper. On top of this, this library has poor documentation and lacks examples other than the official basic client, which also did not help while trying to build the wrapper.

Meanwhile, on the coordination protocol part of the project there were also some issues while developing it using C++, mostly due to their unfamiliarity with the language. Due to both of this issues, we started considering a java implementation and ultimately ended up making a joint decision to try to remake everything using Java. Given how both groups were familiar with the latter, this was considerably faster and it took just a few days to port all the C++ code we developed until then.

4.4.3 The Java implementation

Given how using the Java ROS library would require a system downgrade, we were a bit apprehensive about using Java at first. The most recent version at the time only had official support for ROS Kinetic, but later we ended up trying to run it on ROS Melodic and it worked perfectly. This removed the need for a system downgrade and we were able to end up running it on Ubuntu 18.04.

The Java implementation is built using Java 8 and uses the Curator Framework 4.2 for the communication with the Zookeeper server. The Zookeeper server used is the version 3.5.3 and an instance of it needs to be running before the use of the service.

Given that the final goal is integrating it with other components, we tried to make

the implementation as modular as possible. Another advantage of this approach is easily being able to replace any component of the final system.

Chapter 5

Evaluation and demonstration

In this chapter we will evaluate the implementation by using two different approaches. The first is the functional evaluation, which shows a visual representation of the implementation and the system being used. The second is the performance evaluation, where we test the performance of the membership service by doing load tests as well as testing how long operations take to be completed.

5.1 Functional evaluation

For the Functional evaluation, the simulation built to aid visualize the system is presented, followed by another subsection that shows the details of the created scenario. Then the solution for integration of the membership service with the coordination protocol is also presented, followed by a subsection that shows the details of that solution. Finally, evaluation results are shown.

5.1.1 Simulation scenario

A simple simulation scenario was developed to aid testing and help visualize the whole project working.

There were several requirements to select the simulation software:

- Lightweight in terms of computing resources. This is important because it enables running it on any kind of machine.
- Scalable, to allow big scenarios with support for a modest number of vehicles.
- Communication with ROS, so we can use it as the middleware with the application, as previously explained on chapter 3.
- Support for an easy to use scripting language.
- Good documentation, preferably with multiple examples.

- Adequate for vehicular simulations and/or the versatility to use it for multiple cases.

There are also other important factors that are not absolutely needed but that we took in consideration while choosing, such as:

- Vehicles/road models and textures already available.
- Basic built in path planning capabilities.

Given all the latter requirements, the tool we decided to use is V-REP [24] as it fits all the requirements and preferences. V-REP uses LUA as its scripting language and it allows the addition of scripts to each object that respond to some events and can perform actions on the simulation environment. These scripts can be threaded or non threaded.

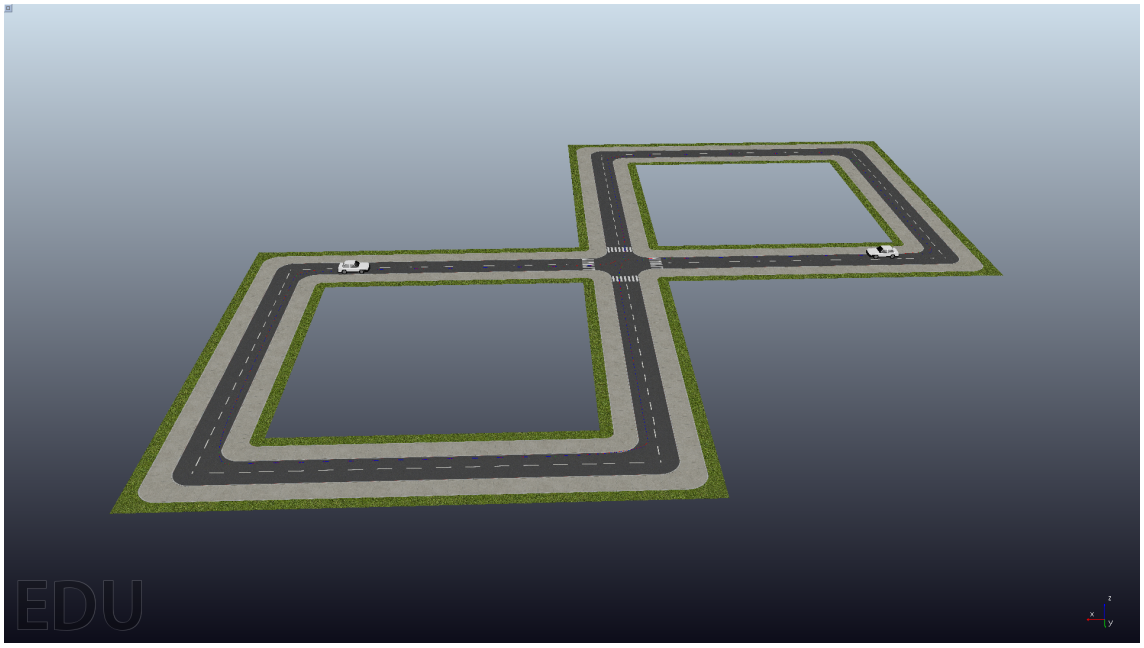

Figure 5.1: Screenshot of the developed simulation scenario

5.1.2 Simulation architecture

We designed a simple scenario with a 8 shaped road with a 4-way intersection and 2 vehicles going in opposite directions towards the intersection. This scenario could be used for multiple settings with any amount of vehicles, so we built it in a way that can be easily modified if necessary, which means that actions like adding/removing vehicles or changing the road should only take a couple of minutes. Figure 5.1 illustrates the final scenario in the simulator.

The simulation uses one threaded script for each vehicle, which are the only scripts used.

Each vehicle is a simple dumb object that does not have sensors or any kind collision logic. The only information we can get about them is their velocity, acceleration, position and state (running/stopped). In terms of controls, we are able to set the velocity and acceleration, as well as send commands to perform some basic actions: start, stop and change color.

Their normal behavior is following a path object that is pre-defined in the simulation. The default path is a simple loop around the whole road with a left-turn at the intersection. This means that both vehicles follow the same path and will eventually start over at their start positions.

Figure 5.2 shows the window used to control each vehicle during the simulation. It has sliders to control the velocity and acceleration of the vehicle, and a label that shows wether the vehicle is running or not.

Figure 5.2: Window used to control each vehicle

Right before each crosswalk there is a specific checkpoint. Upon reaching this checkpoint, the vehicle calls a function in its personal script called `tryGet()`. This function stops the vehicle, changes its color to yellow and publishes a `TRYGET` message in ROS, which means this vehicle wants to perform a maneuver.

When this request is approved, the vehicle that made it starts following the path again and turns green, while every other vehicle associated with the maneuver stops and turns red. These vehicles will eventually receive another command to start following the path again.

The communication between the simulator and the back end is done using ROS via publish/subscribe and two topics are used. The first is `/agentState` in which vehicles periodically publish their current state (position, velocity, acceleration, running flag) and `TRYGET` requests. The second topic is `/agentControl`, in which an external entity publishes commands to control each individual vehicle. The possible controls are exactly the same ones available in the simulation: start, stop and change color. Every vehicle is always subscribed to this topic so they can execute the actions as soon as they are received. Every message is always identified with a vehicle specific UUID, so that published information can be identified and targeted to the right vehicle.

We built a java module to interact with this simulation via ROS using these topics, and we will go thought this module on section 5.1.3.

5.1.3 Integration development

Given that this project is part of a larger system, integration was an important factor for the evaluation as a whole. The first step was to build the simulation that we went through on the previous section, and the second step was to build the back end for that simulator that would fully integrate it with the coordination protocol and the membership client.

The full project was made to be as modular as possible. This way it was easy to add new modules or work on features separately and then being able to connect them later.

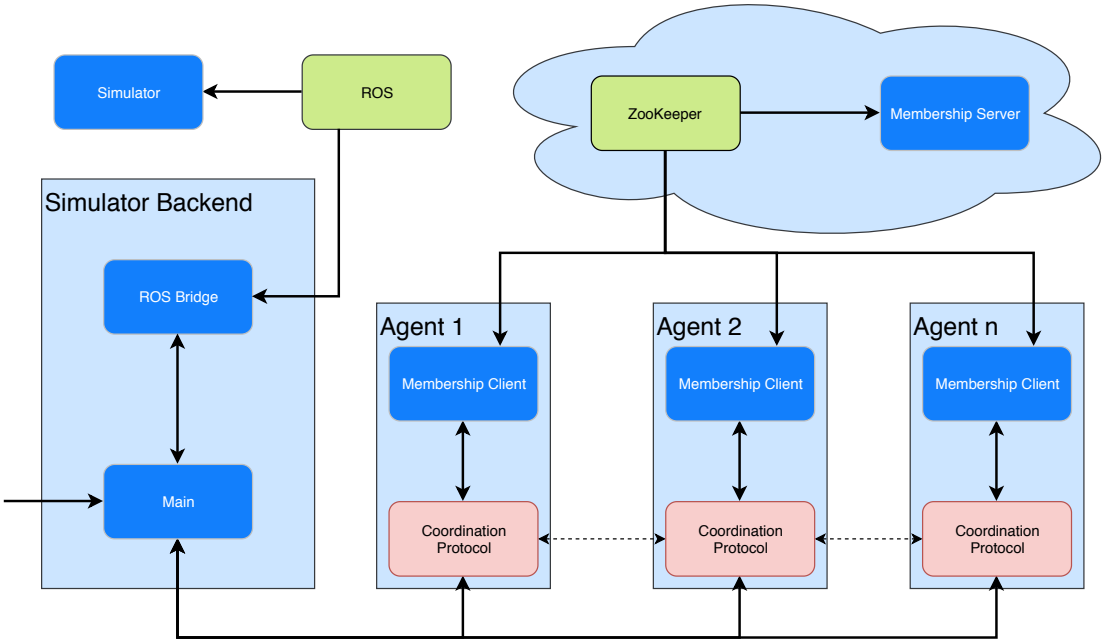

Figure 5.3: Integrated flow diagram with the simulator

Figure 5.3 shows a flow chart of what the integrated solution to use the service with the simulator looks like. The modules in blue correspond to what was done on this dissertation, while meanwhile the other ones are external.

We built a simulator backend, which is a component used to interact with the simulator and create instances of agents considering the data received from the simulation. THe full integrated system works as follows:

1. The simulator publishes the state of its agents frequently to the ROS server.
2. The `ROS Bridge` module inside the Simulator Backend component subscribes to the topic where ROS published the agent state information. This data is periodically picked up by `Main`.

3. As `Main` gets this data, it creates instances of the `Coordination Protocol` for each agent corresponding to the ones existing in the simulator.

4. These `Coordination Protocol` instances constantly update their state on the Membership Service using the `Membership Client`. As we have previously explained, the client pushes that data to the `Zookeeper` to be fetched later on by the `Membership Server`, which pushes the membership information back to `Zookeeper`.

5. When it wants to perform a maneuver, the `Coordination Protocol` asks the `Membership Client` for its Membership, which is pulled from `Zookeeper`.

6. With the membership information, each `Coordination Protocol` can connect to the `Coordination Protocols` of other agents.

7. Whenever `Main` gets a `TRYGET` request, it redirects it to the right Agent, which uses its `Coordination Protocol` to interact with other Agents to coordinate a maneuver.

8. As the `Coordination Protocol` decides what to do, it sends commands to the `Main`, that are redirected to ROS using the `ROS Bridge`.

9. The Simulator picks up these commands from ROS and executes them.

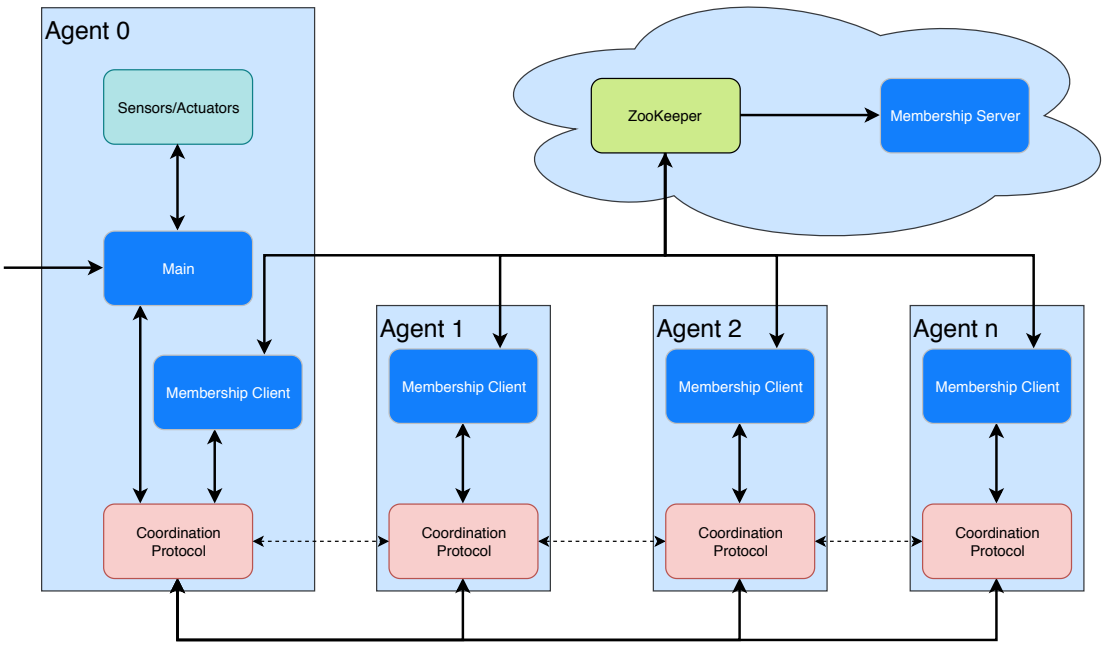

Figure 5.4: Integrated flow diagram in the real world

Figure 5.4 shows exactly the same integration diagram, but using a real world scenario instead of a simulator. As it can be seen in this diagram, the main difference is that the agent back-end does not exist, and instead it is replaced by a simple agent with an entry point to run the application. In this scenario, state information would be sent to the `Main` by the vehicle sensors, and the `Main` would send commands using the vehicle actuators. This `Main` only contains one single instance of the Coordination Protocol, which is used exactly the same way as in the simulator scenario.

Since all the sensors and actuators are inside the vehicle itself, we do not use ROS in this scenario, as we are able to transfer information between this components directly.

5.1.4 Integration architecture

Figure 5.5 shows the same diagram we have seen on Figure 4.2, but now with the new modules added for integration purposes and how they interact with everything else.

The entrance of the program is a module called main. This module is used to execute the program and it links every other module by performing the following steps in this exact order:

1. Initialize an empty map `agentInstances`. This map will store all the agent instances created by the simulator, mapped by their UUID.

2. Create an instance of `TryManeuverCallback`. This interface has a function called `tryManeuver()` that takes an UUID. In this case, the implementation of this interface is using the provided UUID to take the corresponding agent instance from the map created in the previous step, and calling the function of that object with the same name. The idea of this callback is to call it every time a `TRYGET` request is received and redirecting that request to the coordination protocol.

3. Instantiate `AgentManager` giving it the `TryManeuverCallback` created on the previous step. `AgentManager` will initialize the ROS connection, start listening to `/AgentState` and initiate a publisher on `/AgentControl`. This publisher can be used to control the simulator using functions provided by the `AgentControl`. An internal list of `AgentProperties` is kept updated as the manager receives information from ROS .

4. Start an instance of `RosManeuverCallback` and giving it the `AgentManager` created in the previous step. The idea of this callback is to give it to each agent instance, so they can invoque the control functions of the `AgentManager` that allows them to send commands to the simulator.

5. Periodically calling the function `getAllAgentProperties()` of the `AgentManager` and creating one instance of the Coordination Protocol, adding

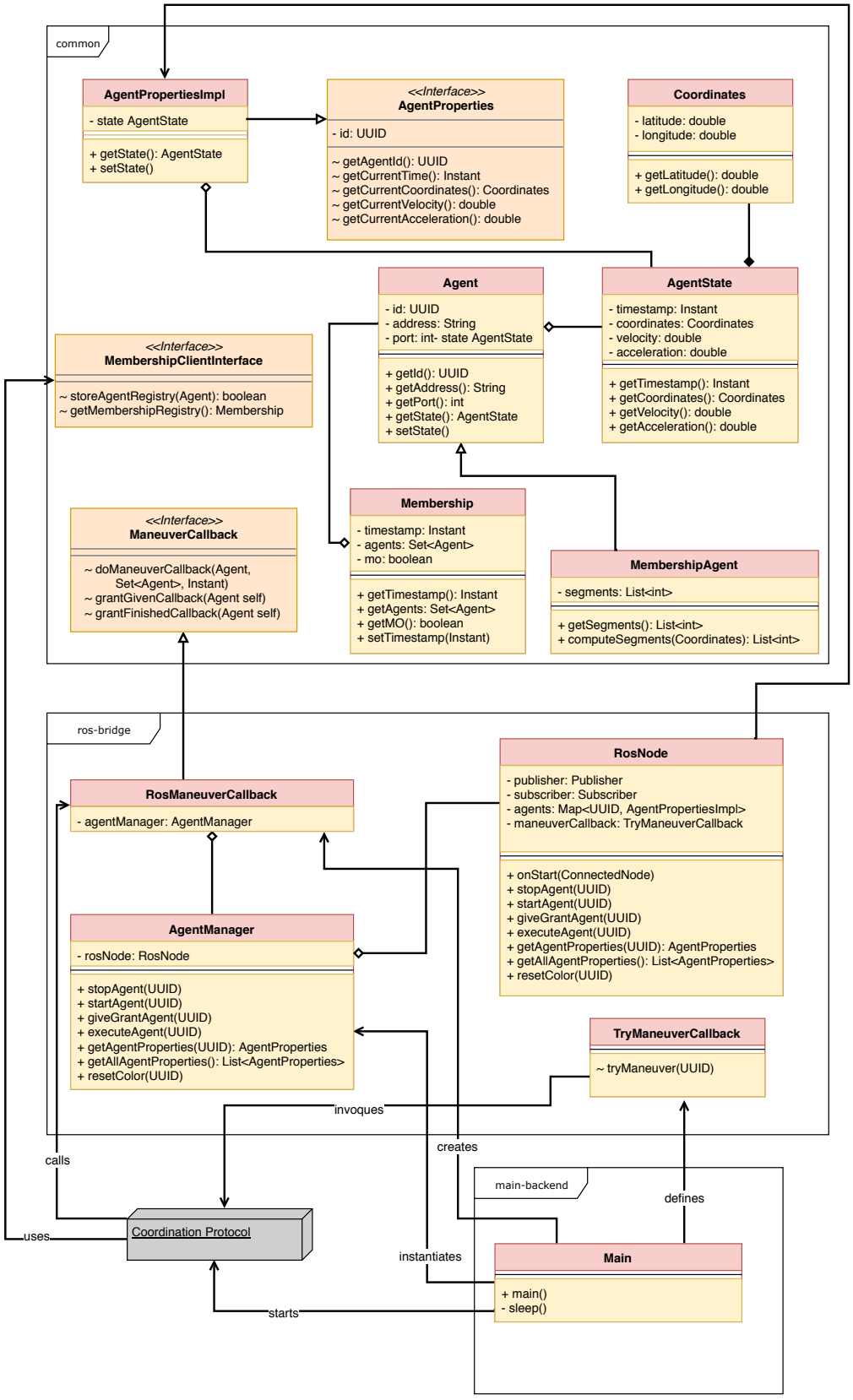

Figure 5.5: Class Diagram with full integration

it to the map created on the first step (if it does not previously exist) while giving the `RosManeuverCallback` to each one of them. These `AgentProperties` objects are mutable and are guaranteed to always be updated in the background as new info is received and processed by the `AgentManager` .

5.1.5 Results and discussion

Only one single scenario was tested, the one we can see in Figure 5.1 that only contains two vehicles.

We started by setting the velocity and acceleration to 1 using the vehicle control window that we can see on Figure 5.2 for both vehicles, with a slightly offset so one of them gets to the stopping point first. Once the first one got to the stopping point, right before the crosswalk, it turned yellow, and both of them stopped. Shortly after, the yellow one turned green and started following the path again, while the other one turned red. After around one second, both vehicles turned gray and kept following their path.

After this first maneuver, we decided to speed up the simulator to see if things kept working. We changed velocity and acceleration to the maximum value, 10 and everytime they reached the stopping point at the crosswalks the same behavior could be seen.

There was a little race condition where the vehicle making the maneuver request would not change its color to green, but this was just a simple graphical bug and it did not affect the performance of the system.

Given this results, we can say that everything worked as expected.

5.2 Performance evaluation

We decided to test the membership service by using three different approaches: processing time, load tests and connection latency.

5.2.1 Processing time

For this test we decided to see how the server processing time is affected by the number of clients using it. It was executed in a single machine with an AMD Ryzen 2700X 8-core 16-threads processor running at 3.8 Ghz.

We tried to take load issues out of the equation while performing this test by making sure the membership server was the only machine connecting to the Zookeeper server. To do this, we created a simple client generator that created a certain number of clients, pushed their information to Zookeeper and slept for a certain amount of time, without closing the Zookeeper connection.

We ran this test for an incrementing number of clients and Figure 5.6 shows the results of this experiment. As we can see, initially there is a linear growth in processing time

that eventually becomes exponential, which is explaining by that fact that to process the membership of each client, the server needs to take in consideration every other vehicle inside the segment.

Later, a similar test was performed with the client module by changing the client generator to periodically make push and pull requests on all clients, and measuring how much time that set of operation takes on average. In this case, we found out that ignoring possible load issues, processing time is exactly the same no matter how many clients are connected to the service and is always around 8 ms. This is explained by the way the service is built, as the client simply writes and reads very specific node paths on Zookeeper, without the need of extra queries.

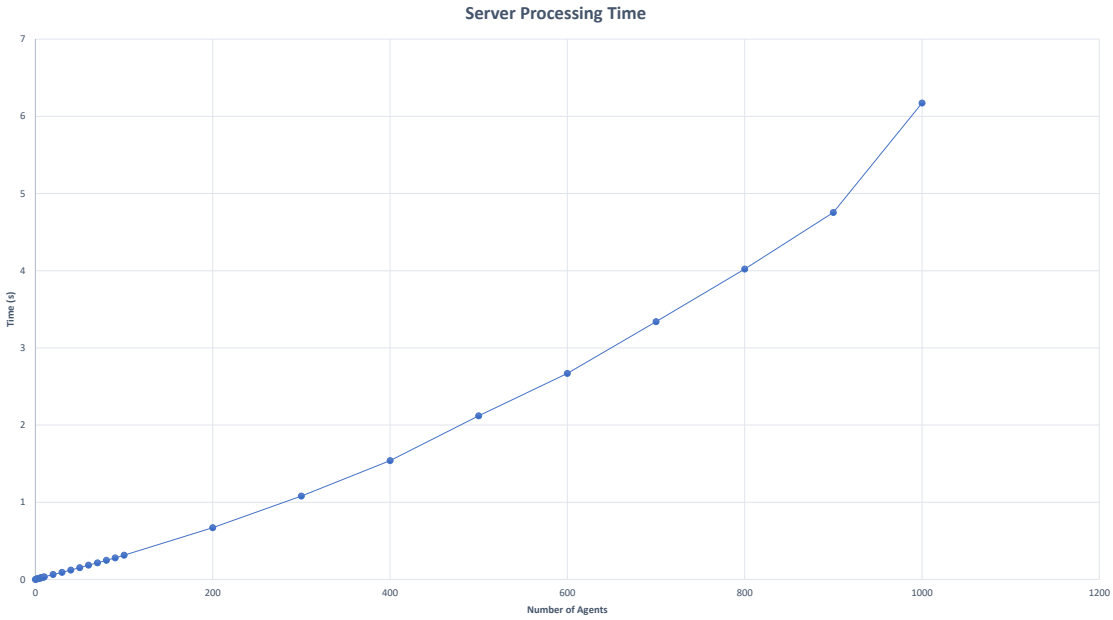

Figure 5.6: Server processing time by number of vehicles

5.2.2 Load tests

Load tests were performed on Amazon Web Services (AWS). We created 7 basic virtual machines with 1 vCPU each, 1 used for the Zookeeper server, another used for the Membership Server, and 5 used for the Clients generators used in the previous test.

These virtual machines are really weak compared to the machine used on the previous test, so we decided to start by checking how many clients each client generator would be able to manage without overloading itself. This simple test shows that each one of them was able to handle up to 60 clients without any performance hit. After that the single vCPU is unable to manage all the threads without slowdowns.

Knowing this, we decided to test it in increments of 10 from 10 up to 60 clients per instance, for a total of 300 clients. We measured the time it takes for a full server cycle

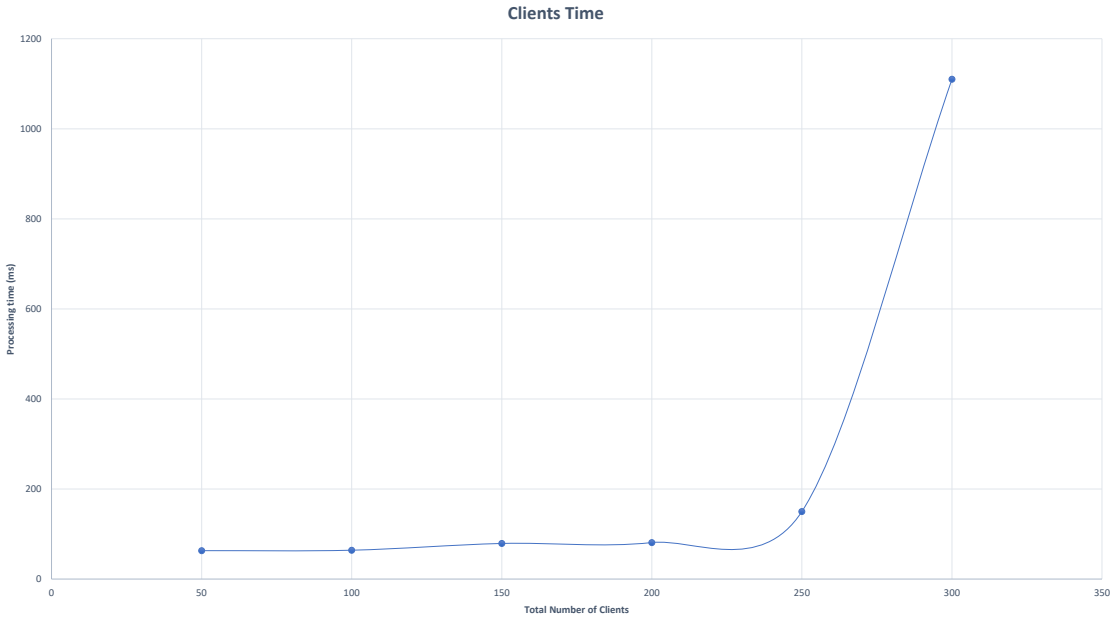

Figure 5.7: Client processing time when overloaded

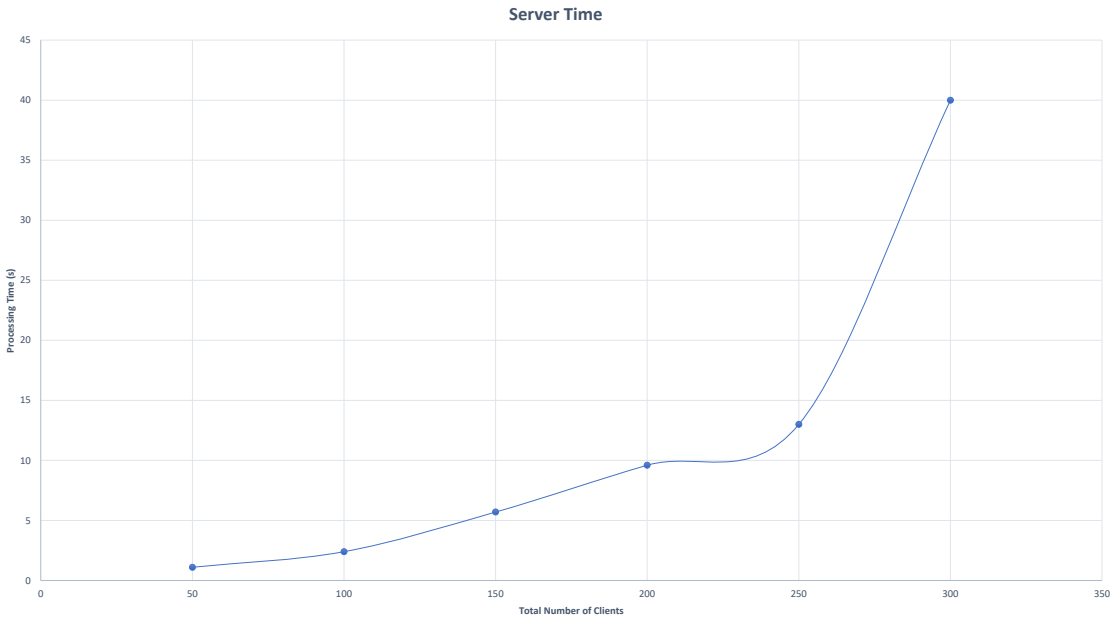

Figure 5.8: Server processing time when overloaded

(pulling agents, processing, pushing membership), as well as a full client cycle (pushing data, pulling membership). The timer used between each client cycle was 100 ms.

Figures 5.7 and 5.8 show the results of this test for both the client and the server.

The server time grows exponentially as seen on the previous example, but we can notice that this time the growth from 250 to 300 is a lot higher than what can bet expected from the previous results.

The client time stays linear at the start, similarly to the previous test, however, we can see that it grows exponentially from 250 to 300, which means the system is being bottlenecked. At this point we could also verify that the clients were starting to throw connection errors, which means that the Zookeeper server was getting overloaded with requests and was unable to answer more connections. While this was happening, we could also verify that after getting the agent data from the Zookeeper server, the membership server would process the membership in the expected time for the incrementing number of clients that we have seen on the previous test, which shows that its performance is not being affected by the increase in clients.

In summary, we can conclude that the Zookeeper server is being the bottleneck of this system. These results can be explained by the fact that this server was not able to handle more connections, and consequently answers were being delayed, as shown by the graphs.

5.2.3 Connection latency

Since latency is an important factor to consider for this system, we decided to perform a small latency test. For this test, a basic virtual machine with 1 vCPU was created on AWS and it was used to run both the Zookeeper server and the Membership server. We ran one simple client in our laboratory that connected to it and measured the time to access the service.

Both server were running on the same machine to minimize the latency time between them. This seems accurate as in a real deployment, both servers would preferably always be running next to each other, even if on different machines.

This virtual machine was running in the East US region, and our Laboratory in Lisbon, which means that the distance was around 6500 Km. For this distance, the client took an average of 700 ms for each full cycle of pushing and pulling information from zookeeper, which means there was around 450 ms of latency for each round-trip operation.

We concluded that distance is an important factor to be taken into consideration for a system like this, and that servers should be deployed close to their clients, or possibly using fog/edge computing as an alternative solution.

Chapter 6

Conclusions and future work

In this chapter some conclusions and future work are presented.

6.1 Conclusions

We implemented a cloud based membership service to aid cooperative vehicles that was previously suggested in [5].

On chapter 2 we presented the state of the art in cloud computing, autonomous and cooperative vehicles as well as existing communication options. This information gave some context about important details that need to be taken into consideration while building a system like this, as well as some insight on what technologies might be used.

On chapter 3 the design of the membership system is presented, starting by introducing the problem it wants to solve, as well as the work it is based on [5]. A membership protocol was designed to use it with, some important features of the system were explained, taking into consideration important issues like scalability.

On chapter 4 the implementation challenges are presented, followed by a detailed view of the middleware solution used, as well as the full software architecture. Finally, the solution implementation was described, and some of the possible approaches and decisions taken along the way were mentioned, as well as details about the final implementation.

On chapter 5 two types of evaluation were performed. The first was the functional evaluation, which consisted in constructing a visual representation of the service using a simulator, as well as integrating it with a coordination protocol. At the end of this section, results are described, which show that everything works as expected and there is a a good graphical representation of the system being used. The second was the performance evaluation, where the performance of the membership service was measured by analyzing processing and latency times, as well as doing load tests to discover the bottleneck of the system, which we ended up concluding that it is the Zookeeper server.

As it can be seen on this last chapter, it is possible to conclude that the final solution works well but does not have the expected performance for a real world usage given how

big the delays are when one single server takes care of multiple clients. However, there are multiple solutions to this issue, and some of them will be mentioned in the next section.

6.2 Future work

The following aspects has been left as future work:

We aim at adding a predictive algorithm. The predictive functions currently return every vehicle inside the segment, which is obviously not the expected behavior in the final product. A good implementation could use machine learning techniques to make proper predictions.

We aim at making the current implementation map aware. In the future the segment mapping should be done using a real world map as a base to translate positions into segments. Having map data can also help understanding exactly what needs to happen while performing a maneuver, which also helps the predictive algorithm.

We aim at addressing security issues, hence relaxing one of our previous assumptions. With the current implementation, a malicious user can disturb the system in any way it wants since currently communication between the membership client/server and the Zookeeper server is not secure, is are completely trust based.

Finally, the current performance is overall not adequate for a real world scenario since the predictions take way too long when the membership server needs to handle a considerable about of vehicles. Using a Zookeeper cluster, addressing the bottleneck problem identified, should obviously help with performance, but it is still far from ideal. Zookeeper is a great tool, but this is probably not one of the best uses for it, given that its best use case is one with high reads and low writes, which is very different from our needs.

We aim at addressing the bottleneck problem. Two possible solutions exist, namely, using multiple Zookeeper clusters or replacing Zookeeper with some other type of communication. For this second approach we suggest using direct communication and creating an intermediate database server prepared for a high number of both read and write requests. This application could also have function calls that return targeted data personalized for whoever made request, which should be able to make communication faster by reducing the amount of requests needed. An approach like this should keep most of the advantages of using Zookeeper while also providing better performance.

Glossary

AWS Amazon Web Services. 39, 41

CC Cloud Computing. 5

IoT Internet of Things. 12

MANET Mobile ad hoc network. 11, 12

NIST National Institute of Standards and Technology. 5

P2P Peer to Peer. 13

ROS Robot Operating System. 23, 31, 33, 34, 36

SAE International Society of Automotive Engineers. 7

TTL Time to Live. 25

UUID Universally Unique Identifier. 24, 33, 36

V2I Vehicle to Infrastructure. 2, 11

V2R Vehicle to Road. 13

V2V Vehicle to Vehicle. 2, 6, 10, 11, 13, 18

VANET Vehicular ad hoc network. 12, 13

VM Virtual Machine. 5

References

[1] United States Environmental Protetion Agency. Global Greenhouse Gas Emissions Data. `https://www.epa.gov/ghgemissions/global-greenhouse-gas-emissions-data`. [Online; accessed 04-December-2018].

[2] Soyoung Ahn and Michael J Cassidy. Freeway traffic oscillations and vehicle lane-change maneuvers. In *Transportation and Traffic Theory 2007. Papers Selected for Presentation at ISTTT17Engineering and Physical Sciences Research Council (Great Britain) Rees Jeffreys Road FundTransport Research FoundationTMS ConsultancyOve Arup and Partners, Hong KongTransportation Planning (International) PTV AG*, 2007.

[3] J. G. Andrews, S. Buzzi, W. Choi, S. V. Hanly, A. Lozano, A. C. K. Soong, and J. C. Zhang. What will 5g be? *IEEE Journal on Selected Areas in Communications*, 32(6):1065–1082, June 2014.

[4] Carl Bergenhem, Steven Shladover, Erik Coelingh, Christoffer Englund, and Sadayuki Tsugawa. Overview of platooning systems. In *Proceedings of the 19th ITS World Congress, Oct 22-26, Vienna, Austria (2012)*, 2012.

[5] Antonio Casimiro and Elad M. Schiller. Membership-based maneuver negotiation in safety-critical vehicular systems. In *Technical report*, Chalmers University of Technology, Department of Computer Science and Engineering, March 2018.

[6] Gilbert Chlewicki. New interchange and intersection designs: The synchronized split-phasing intersection and the diverging diamond interchange. In *2nd Urban Street Symposium, Anaheim, California*, pages 28–30, 2003.

[7] S. Corson and J. Macker. Mobile ad hoc networking (manet): Routing protocol performance issues and evaluation considerations, 1999.

[8] A. Das, I. Gupta, and A. Motivala. Swim: scalable weakly-consistent infection-style process group membership protocol. In *Proceedings International Conference on Dependable Systems and Networks*, pages 303–312, June 2002.

[9] Luke Dormehl. Elon Musk thinks Starlink satellite internet could be online before 2021. https://www.digitaltrends.com/cool-tech/starlink-working-sooner-than-you-expect/. [Online; accessed 18-June-2019].

[10] The Economist. The great crawl. https://www.economist.com/china/2016/06/16/the-great-crawl, 2016. [Online; accessed 29-November-2018].

[11] Michel Ferreira, Ricardo Fernandes, Hugo Conceição, Wantanee Viriyasitavat, and Ozan K. Tonguz. Self-organized traffic control. In *Proceedings of the Seventh ACM International Workshop on VehiculAr InterNETworking*, VANET '10, pages 85–90, New York, NY, USA, 2010. ACM.

[12] United Kingdom Department for Transport. Reported Road Casualties Great Britain: 2014 Annual Report: Contributing Factors to Reported Road Accidents 2014. https://www.gov.uk/government/uploads/system/uploads/attachment_data/file/463043/rrcgb2014-02.pdf. [Online; accessed 06-December-2018].

[13] A. Ghosh, R. Ratasuk, B. Mondal, N. Mangalvedhe, and T. Thomas. Lte-advanced: next-generation wireless broadband technology [invited paper]. *IEEE Wireless Communications*, 17(3):10–22, June 2010.

[14] Mark Handley. Delay is not an option: Low latency routing in space. In *Proceedings of the 17th ACM Workshop on Hot Topics in Networks*, HotNets '18, pages 85–91, New York, NY, USA, 2018. ACM.

[15] Red Hat. JGroups - A Toolkit for Reliable Messaging. http://www.jgroups.org/overview.html. [Online; accessed 12-December-2018].

[16] Patrick Hunt, Mahadev Konar, Flavio P. Junqueira, and Benjamin Reed. Zookeeper: Wait-free coordination for internet-scale systems. In *Proceedings of the 2010 USENIX Conference on USENIX Annual Technical Conference*, USENIXATC'10, pages 11–11, Berkeley, CA, USA, 2010. USENIX Association.

[17] IEEE StandardIEEE Standards Association. Part 11: Wireless lan medium access control (mac) and physical layer (phy) specifications amendment 6: Wireless access in vehicular environments. Standard IEEE 802.11p-2010, IEEE Standard for Information technology, 2010.

[18] SAE International. Taxonomy and definitions for terms related to driving automation systems for on-road motor vehicles (2018). http://standards.sae.org/j3016_201806/. [Online; accessed 07-December-2018].

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

[20] Douglass Lee Jr, Lisa Klein, and Gregorio Camus. Induced traffic and induced demand. *Transportation Research Record: Journal of the Transportation Research Board*, (1659):68–75, 1999.

[21] Peter M. Mell and Timothy Grance. Sp 800-145. the nist definition of cloud computing. Technical report, Gaithersburg, MD, United States, 2011.

[22] Radiocommunication Sector of International Telecomunication Union. IMT Vision - Framework and overall objectives of the future development of IMT for 2020 and beyond. `https://www.itu.int/dms_pubrec/itu-r/rec/m/R-REC-M.2083-0-201509-I!!PDF-E.pdf`. [Online; accessed 11-December-2018].

[23] Stephan Olariu, Tihomir Hristov, and Gongjun Yan. *The Next Paradigm Shift: From Vehicular Networks to Vehicular Clouds*, chapter 19, pages 645–700. John Wiley and Sons, Ltd, 2013.

[24] E. Rohmer, S. P. N. Singh, and M. Freese. V-rep: a versatile and scalable robot simulation framework. In *Proc. of The International Conference on Intelligent Robots and Systems (IROS)*, 2013.

[25] Popular Science Stan Horaczek. Here's where your new car lands on the self-driving scale. `https://www.popsci.com/self-driving-car-scale`. [Online; accessed 13-December-2018].

[26] VSAT Systems. Latency- why is it a big deal for Satellite Internet? `http://www.vsat-systems.com/satellite-internet-explained/latency.html`. [Online; accessed 11-December-2018].

[27] Tesla. Tesla Autopilot. `https://www.tesla.com/autopilot`. [Online; accessed 13-December-2018].

[28] Md Whaiduzzaman, Mehdi Sookhak, Abdullah Gani, and Rajkumar Buyya. A survey on vehicular cloud computing. *Journal of Network and Computer Applications*, 40:325 – 344, 2014.

[29] R. Yu, Y. Zhang, S. Gjessing, W. Xia, and K. Yang. Toward cloud-based vehicular networks with efficient resource management. *IEEE Network*, 27(5):48–55, September 2013.

[30] Sherali Zeadally, Ray Hunt, Yuh-Shyan Chen, Angela Irwin, and Aamir Hassan. Vehicular ad hoc networks (vanets): status, results, and challenges. *Telecommunication Systems*, 50(4):217–241, Aug 2012.