CISTER
Research Centre in
Real-Time & Embedded
Computing Systems

BEng Thesis

Towards Wireless ADAS - Retrofitting IoT for Increased Safety

Orientação científica: Ricardo Severino, Coorientação: Harrison Kurunathan

Tiago Pinto

CISTER-TR-191207

2019/09/11

Towards Wireless ADAS - Retrofitting IoT for Increased Safety

Tiago Pinto

CISTER Research Centre

Polytechnic Institute of Porto (ISEP P.Porto)

Rua Dr. António Bernardino de Almeida, 431

4200-072 Porto

Portugal

Tel.: +351.22.8340509, Fax: +351.22.8321159

E-mail: tiagoamaralpinto@gmail.com

https://www.cister-labs.pt

Abstract

Opening a new world of possibilities, Wireless Sensor Networks (WSNs) are themain reason why the Internet of Things (IoT) is growing at an unstoppable pace.The extension of this recent technological paradigm is so present in our everyday life that it is estimated that more than 75 billion devices are going to beconnected by the year of 2025, a fivefold increase in ten years.This is due to research and industry interests, which collide in a unique philosophy:creating new tools and technologies to support this unprecedented growth.Therefore, the investment in technological interests such as vehicle assistance andautonomous driving increases more and more.However, implementing these same systems on older cars is costly as it wouldbe necessary to redesign its entire structure. This thesis aims to provide resultsand conclusions concerning the supported tools for retrofitting these AdvancedDriver-Assistance Systems (ADAS) via wireless in all the vehicles, increasingsafety road guarantees and V2V (Vehicle to Vehicle) communications, transformingthe vehicles into IoT devices.To support this new paradigm, there is a wide range of wireless communicationprotocols for similar applications. The IEEE 802.15.4 stands out for the useof Time Division Multiple Access (TDMA) in MAC behaviors such as Deterministicand Synchronous Multichannel Extension (DSME), Low Latency DeterministicNetwork (LLDN) and Time Slotted Channel Hopping (TSCH), which weredesigned to support time critical applications.Moreover, to reinforce the assurance of similar critical systems, middlewaresare crucial to provide safety guarantees. As a result, in this thesis, we discuss theperformance of both versions of the Robot Operating System (ROS) as middlewareand the behaviour of the network when supported by them.Consequently, we created a simulation ADAS scenario, which allows us to testthe QoS provided by the IEEE 802.15.4 DSME MAC behavior when supportingADAS and the respective application impact, focusing on the safety guaranteesof these systems.

Towards Wireless ADAS - Retrofitting IoT for Increased Safety

Tiago Manuel Amaral Pinto

Licenciatura em Engenharia Eletrotécnica e de Computadores

Departamento de Engenharia Eletrotécnica
Instituto Superior de Engenharia do Porto
2019

Este relatório satisfaz, parcialmente, os requisitos que constam da Ficha de Unidade Curricular de Projeto/Estágio, do 3º ano, da Licenciatura em Engenharia Eletrotécnica e de Computadores

Candidato: Tiago Manuel Amaral Pinto, Nº 1160938, 1160938@isep.ipp.pt
Orientação científica: Dr. Ricardo Augusto Rodrigues da Silva Severino (PhD), rarss@isep.ipp.pt
Co-orientação: John Harrison Kurunathan (MSc), hhkur@isep.ipp.pt

Licenciatura em Engenharia Eletrotécnica e de Computadores
Departamento de Engenharia Eletrotécnica
Instituto Superior de Engenharia do Porto
5 Setembro 2019

Acknowledgments

Firstly, I would like to thank my parents for all the support, encouragement and accompaniment during this journey.

Secondly, and most of all, I would like to express my deepest appreciation to professor Dr. Ricardo Severino, for his assistance, guidance and patience throughout of the process of developing and writing this thesis. Moreover, I would like to thank my co-supervisor Harrison Kurunathan and my colleague Bruno Vieira for all the help and availability when I needed the most. Without them it would not be possible to achieve this goal.

I would also like to thank all my colleagues at CISTER, who welcomed me and provided as much help as funny moments. As promised, Nuno, Daniel, Rita, Rafael, João, Joel, Renato, and of course, Francisco: thank you for everything.

Last, but not least, I would like to thank Maria, who, without even realizing it, gave me the strength to achieve this milestone. Words are not enough to express my gratitude to you.

Resumo

Abrindo um novo mundo de possibilidades, as Wireless Sensor Networks (WSNs) são a principal razão pela qual a Internet of Things (IoT) está atualmente a crescer num ritmo imparável. A extensão desse recente paradigma tecnológico está tão presente na nossa vida quotidiana que estima-se que mais de 75 bilhões de dispositivos serão conectados até 2025, um aumento de cinco vezes em dez anos.

Isto deve-se aos interesses dos setores de investigação e industriais, que colidem numa filosofia única: criar novas ferramentas e tecnologias para apoiar esse crescimento sem precedentes. Portanto, o investimento em interesses tecnológicos, como assistência a veículos e condução autónoma, aumenta cada vez mais.

No entanto, implementar esses mesmos sistemas em carros mais antigos é caro, pois seria necessário redesenhar toda a sua estrutura. Esta tese visa fornecer resultados e conclusões sobre as ferramentas suportadas para a modernização desses sistemas avançados de assistência ao condutor (ADAS - Advanced Driver-Assistance Systems) via wireless em todos os veículos, aumentando as garantias de segurança nas estradas e as comunicações V2V (Veículo para veículo), transformando os veículos em dispositivos IoT.

Para suportar este novo paradigma, existe uma ampla gama de protocolos de comunicação sem fio para aplicações semelhantes. O IEEE 802.15.4 destaca-se pelo uso do TDMA (Time Division Multiple Access) em comportamentos MAC, como Deterministic and Synchronous Multichannel Extension (DSME), Low Latency Determin-istic Network (LLDN) e Time Slotted Channel Hopping (TSCH), projetado para suportar sistemas críticos.

Além disso, para reforçar a garantia de sistemas críticos semelhantes, os middlewares são cruciais para fornecer garantias de segurança. Como resultado, nesta tese, discutimos o desempenho de ambas as versões do ROS (Robot Operating System) como middleware, e, o comportamento da rede quando suportado por estes.

Consequentemente, criamos um cenário ADAS de simulação, que nos permite testar a QoS (qualidade de serviço) fornecida pelo comportamento do IEEE 802.15.4 DSME MAC ao oferecer suporte ao ADAS e ao respetivo impacto da aplicação, com foco nas garantias de segurança desses sistemas.

Palavras-chave: Wireless Sensor Networks, Internet of Things, Advanced Driver-Assistance Systems, DSME, Robot Operating System.

Abstract

Opening a new world of possibilities, Wireless Sensor Networks (WSNs) are the main reason why the Internet of Things (IoT) is growing at an unstoppable pace. The extension of this recent technological paradigm is so present in our every day life that it is estimated that more than 75 billion devices are going to be connected by the year of 2025, a fivefold increase in ten years.

This is due to research and industry interests, which collide in a unique philosophy: creating new tools and technologies to support this unprecedented growth. Therefore, the investment in technological interests such as vehicle assistance and autonomous driving increases more and more.

However, implementing these same systems on older cars is costly as it would be necessary to redesign its entire structure. This thesis aims to provide results and conclusions concerning the supported tools for retrofitting these Advanced Driver-Assistance Systems (ADAS) via wireless in all the vehicles, increasing safety road guarantees and V2V (Vehicle to Vehicle) communications, transforming the vehicles into IoT devices.

To support this new paradigm, there is a wide range of wireless communication protocols for similar applications. The IEEE 802.15.4 stands out for the use of Time Division Multiple Access (TDMA) in MAC behaviors such as Deterministic and Synchronous Multichannel Extension (DSME), Low Latency Deterministic Network (LLDN) and Time Slotted Channel Hopping (TSCH), which were designed to support time critical applications.

Moreover, to reinforce the assurance of similar critical systems, middlewares are crucial to provide safety guarantees. As a result, in this thesis, we discuss the performance of both versions of the Robot Operating System (ROS) as middleware and the behaviour of the network when supported by them.

Consequently, we created a simulation ADAS scenario, which allows us to test the QoS provided by the IEEE 802.15.4 DSME MAC behavior when supporting ADAS and the respective application impact, focusing on the safety guarantees of these systems.

Keywords: Wireless Sensor Networks, Internet of Things, Advanced Driver-Assistance Systems, DSME, Robot Operating System.

Contents

List of Figures

List of Tables

List of Acronyms

Acronym	Description
ACK	Acknowledgment
ACP	Allocation Contention Periods
ADAS	Advanced Driver-Assistance Systems
API	Application Programming Interfaces
BE	Beacon enabled
BI	Beacon Interval
BO	Beacon Order
CAP	Contention Access Period
CCA	Clear channel Assessment
CFP	Contention Free Period
CSL	Coordinates Sampled Listening
CSMA-CA	Carrier Sense Multiple Access with Collision Avoidance
DDS	Data Distribution Service
DSME	Deterministicand Synchronous Multichannel Extension
DSME-GTS	DSME Guaranteed Time Slot
EB	Enhanced Beacon
FFD	Fully Function Devices
GACK	Group Acknowledgment
GDS	Global data Space
GTS	Guaranteed Time Slots
GUI	Graphical User Interface
IDL	Interface Definition Language
IoT	Internet of Things
LLDN	Low Latency Deterministic Network
MAC	Medium Access Control
NBE	Non-Beacon Enabled
P2P	Peer-to-Peer
PAN	Personal Area Network
PHY	Physical Layer

Acronym	Description
PLCP	Physical Layer Convergence Protocol
PNP	Permission Notification Periods
QoS	Quality of Service
RFD	Reduced Function Devices
RIT	Receiver Initiated Transmissions
ROS	Robot Operating System
RTOS	Real Time Operative System
RTPS	Real-Time Publish-Subscribe
SAP	Service Access Point
SD	Superframe Duration
SO	Superframe Order
SONAR	Sound Navigation and Ranging
SUV	Sport Utility Vehicle
TSCH	Time Slotted Channel Hopping
TDMA	Time Division Multiple Access
URDF	Unified Robot Description Format
URL	Uniform Resource Identier
V2V	Vehicle to Vehicle
WLAN	Wireless Local Area Network
WSN	Wireless Sensor Networks

Chapter 1

Introduction

1.1 Overview

Currently, in modern societies, connectivity, information and security are three of the most relevant subjects. We are witnessing a time in history where mankind is achieving goals that were thought to be unreachable in the last decades. Self-driving cars and smart homes are no longer only a reality in the sci-fi movies. Collect and interconnect data from several devices all over the world is the "heart" of this technological advances. And the trend is that this unprecedented share of information worldwide continues to grow.

This recent technological paradigm known as Internet of Things (IoT) enables a global network infrastructure based on standards and protocols in which devices can interact with each other through smart interfaces integrated in a network of information [13].

According to the World Health Organization [14], approximately 1.35 million people die each year as a result of road traffic crashes. Most of these are due to human negligence, fatigue or lack of safety in older vehicles. In order to decrease these number in the coming years, the incorporation of devices capable of creating an interface between the vehicle and the driver is mandatory. These Advanced Driver-Assistance Systems (ADAS) aims to be one of the biggest contribution in terms of human safety, from an IoT perspective.

One of the many challenges of this concept is the introduction of pre-existing applications in the global network. Applications such as vehicles would need to re-design their infrastructure in order to implement a sensor and wire based communication system.

Fortunately, it is possible to ensure this concept on a wireless communication basis. Wireless Sensor Networks (WSNs) enables a wide bandwagon of possi-

bilities to be implemented in real-time applications [15], increasing, therefore, human safety in a wide range of different areas, such as the automotive sector. This technological concept allows the retrofitting of these ADAS in older vehicles.

Regarding the requirements of ADAS, it is necessary that the monitoring system fulfill the best quality of service (QoS), by using different methods based on a redistribution of the available bandwidth between the various types of traffic [16]. Therefore, to meet these high quality demands that ADAS requires, the IEEE 802.15.4 standard provides several MAC behaviors that are designed to support multi-channel frequency hopping mechanism, such as the Deterministic and Synchronous Multichannel Extension (DSME), Time Slotted Channel Hopping (TSCH) and Low Latency Deterministic Network (LLDN), which uses Time Division Multiple Access (TDMA) to provide timing guarantees [15].

Naturally, to increase robustness, middleware based systems guarantees fault tolerance and error handling. Moreover, allows heterogeneous systems to work together with accessible interfaces with QoS provider. For the IoT domain (sensors and actuators), the Robot Operating System (ROS) is the "holy grail" of application middlewares. The ROS provides low-level device control, hardware abstraction, implementation of commonly-used functionality and message-passing between processes [17].

Although ROS seems to be the obvious choice for ADAS, it does not satisfy real-time run requirements. Despite the fact that ROS provides a good QoS when supporting a wire sensor network, applications that demands high reliability regarding time critical factors cannot rely on this framework since it cannot guarantee fault-tolerance, deadlines, or process synchronization [18]. Additionally, ROS uses TCP as the underlying transport, which is unsuitable for lossy networks such as wireless links. Thus, ROS it not suitable for ADAS.

To overcome this issue, the ROS community works in a continually way to upgrade this open-source middleware to a new version (ROS2) that includes use cases capable of satisfying demands such as real-time systems and small embedded platforms. Moreover, ROS2 can support different operating systems (OS) such as Real-Time OS (RTOS). Therefore, the ROS transport system was replaced by a new one, named Data Distribution Service (DDS), that is able to provide a reliable publish/subscribe transport similar to the older version and solutions for some real-time environments. DDS uses UDP as its transport, which gives control over the level or reliability a node can expect and act accordingly.

Therefore, in this thesis, we created a simulation ADAS scenario based on both ROS versions, which is supported by the IEEE 802.15.4 DSME MAC behavior, in order to take conclusive results regarding the safety guarantees on the implementation of wireless ADAS in older vehicles.

1.2 Research Context

This thesis is being carried out at CISTER Research Centre in Real-Time Embedded Computing Systems, within the context of the ICARUS interest group (Interest group on Cooperative Autonomous Reliable Systems), which is a research framework aiming at improving the safety and security of autonomous cooperative systems by relying on safer digital communications technologies such as the IEEE 802.15.4 protocol to support safety critical IoT wireless applications.

Inside this new paradigm, the automotive sector stands out for its opportunity to take advantage of this technological leap to increase the safety and reliability of existing vehicles, thereby reducing the cost that usually is associated within similar ADAS implementations in most recent high-end vehicles, by exploring a retrofitting strategy of ADAS to all vehicles.

1.3 Research Objectives

The main objective of this thesis is to evaluate the network and middleware performance of a ADAS system supported by a WSN. To achieve decisive results, we propose the design and implementation of an ADAS simulation scenario, which must contain sensors that must be supported by both versions of ROS as middleware and a WSN for device communications, which takes advantage of MAC behaviors provided by the 802.15.4e protocol in order to provide time critical guarantees. This thesis also aims to provide the best configurations for the DSME MAC behavior when supporting similar safety critical systems as ADAS.

1.4 Research Contributions

The main research contributions of this thesis are:

- Implementation support of a co-simulation environment that joins ROS/Gazebo with a network simulator using the DSME communications stack.
- Implementation of an evaluation scenario for analysing the performance of a wireless ADAS system using 802.15.4 DSME communication infrastructure.
- Performance Analysis of a Wireless ADAS scenario, particularly focusing on the impact of several different network settings on the application.

This contribution allows to build a bridge between the research and industrial interests, in which we provide security guarantees regarding the use of these groundbreaking technologies that, due to their recent release and constant upgrades, are not very used in the industry or in any other sector.

1.5 Structure of this Thesis

This thesis is composed of six chapters, including the current one. The following chapter introduces the IEEE 802.15.4 protocol with a closer look into the three of the five MAC behaviors that allows the implementation in time critical applications, namely DSME, LLDN and TSCH. The third chapter presents and describes the most important tools and technologies used throughout this thesis, apart from the standard. Chapter 4 presents the system architecture of our specific use case simulation, in which the results are shown in Chapter 5. Last, the Chapter 6 states the conclusions obtained in the work developed, along with an overview of the future work required to accomplish this thesis objectives.

Chapter 2

Overview of IEEE 802.15.4

This chapter presents and describes the most important features of the IEEE 802.15.4. which is the protocol supported by the Wireless Sensor Network in the Advanced Driver-Assistance System scenario.

2.1 Overview

Growing a fast pace, Wireless Sensor Networks (WSNs) play a key role regarding the evolution of the Internet of Things (IoT) since they represent the main communication infrastructure through which any computational system can interact with the physical world [19]. As a result, there is a wide range of wireless communication protocols for each target application.

As sensor nodes are generally battery-powered devices, the critical aspects to concern are related to reducing the energy consumption of nodes, so that the network lifetime can be extended to reasonable times [20].

Efficiency is thus, a main concern when designing a WSN. However, in certain applications such as critical domains, there's also additional requirements such as reliability, timeliness and scalability that need to be considered as well. Therefore, WSNs protocols should aim for a meeting point between QoS performance and energy efficiency [21].

As a solution to this new paradigm, IEEE 802.15.4 was published in 2003 for Wireless Personal Area Networks (WPAN) [15]. This standard defines the Physical and MAC (Medium Access Control) layers of the protocol stack and is considered the reference standard for commercial WSNs.

2.2 The Low-Rate WPAN Standard

2.2.1 Fundamental concepts

Low-Rate WPAN is a simple and cheap communication network that allows wireless connectivity in applications with limited power and relaxed throughput requirements. They are able to over-the-air data rates from 20 kb/s to 250kb/s, allocate 16 or 64-bit short extended addresses, Carry Sense Multiple Access with Collision Avoidance (CSMA-CA) channel access and have 16 channels in the 2450 MHz band, 30 channels in the 915 MHz band, and 3 channels in the 868 MHz band [22].

In the IEEE 802.15.4 standard, devices can be classified into Fully Function Devices (FFD) and Reduced Function Devices (RFD) [15]. FFD can operate as the Personal Area Network (PAN) coordinator: the principal controller of the PAN. This device identifies its own network as well at its configurations, in which other devices may be associated [23]. Sometimes, a FFD can also operate as a coordinator, which provides local synchronization services and routing to its neighbors. Every coordinator must be associated to a PAN coordinator and it form its own network if it does not find other networks in its vicinity [15]. In case of not full-filling any of the previous functionalities, the FFD works as an end device.

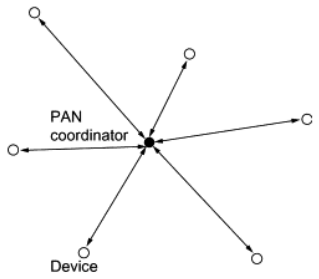

Figure 2.1: IEEE 802.15.4 device communication [1]

The RFD works with the minimal implementation of an end device and of the IEEE 802.15.4 protocol [23]. They are usually suitable for extremely simple applications, since they do not have the need of sending large amounts of data. A FFD can communicate to RFDs or other FFDs, while a RFD can only communicate to a FFD.

The communication between these devices settles on two different topologies: a star topology and a peer-to-peer (P2P) topology. In P2P, each FFD is capable of communicating with other device within its radio range. One FFD acts as the PAN coordinator, and the others FFDs act as routers or end devices to form a

multi-hop network.

In the star topology, one FFD is the PAN coordinator and its located in the star center. All the other FFDs and RFDs behave as generic devices and can only communicate with the coordinator, which synchronizes all the communications in the network. Different stars operating in the same area have different PAN identifiers and operate independently of each other [24].

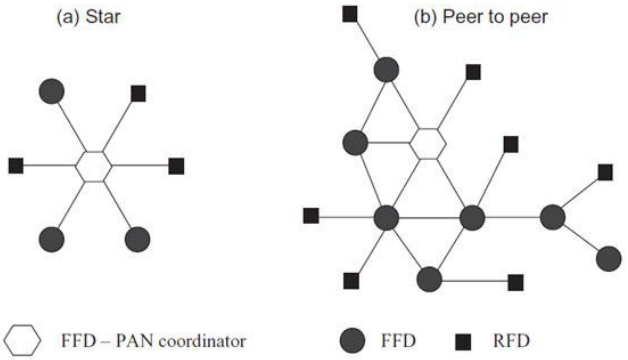

Figure 2.2: Device communication topologies

The IEEE 802.15.4 architecture settles on the Open Systems Interconnection (OSI) seven-layer model. Each layer is in charge of a portion of the standard and offers services to the higher layers.

2.2.2 Architecture

A Lower-Rate WPAN device comprises a Physical layer (PHY), which contains the Radio Frequency (RF) transceiver, and a MAC sub-layer that provides access to the physical channel [22].

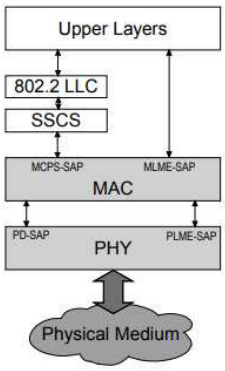

Figure 2.3: IEEE 802.15.4 layer architecture

In IEEE 802.2 protocol structure, Logical Link Control (LLC) was designed to access the MAC sub-layer through the Service-Specific Convergence Sub-Layer (SSCS).

2.2.2.1 PHY

IEEE 802.15.4 operates in three different bands. As depicted in Figure 2.4, it has 16 channels with 2.4GHz and a data rate of 250Kbps, 10 channels with 915 MHz/40Kbps and a single channel with 868MHz and 20Kbps of data rate [15].

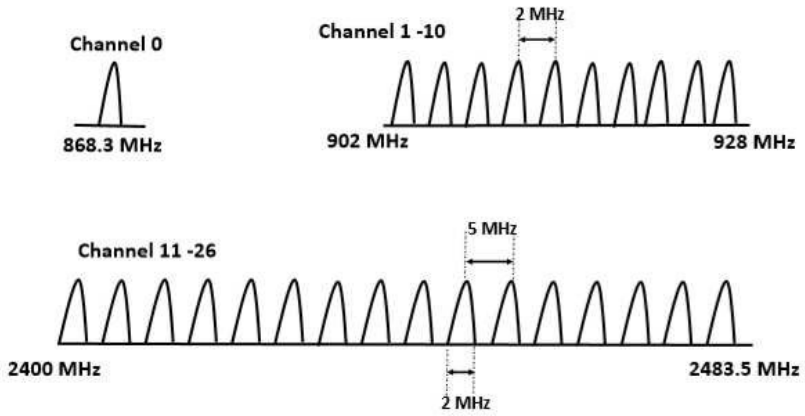

Figure 2.4: IEEE 802.15.4 channel band operations [2]

The MAC layer communicates with the Physical Layer Convergence Protocol (PLCP) sub-layer via primitives through a Service Access Point (SAP) [24].

To perform PLCP functions, the standard specifies the use of state machines that are able to perform Transmission (Tx), Receiving (Rx) and Carrier Sensing (CS) functions. Both MAC and PHY conceptually include management entities, referred to as the MAC Layer Management Entity (MLME), and the Physical Layer Management Entity (PLME) [24].

By taking advantage of PLCP functions, the PHY is where many crucial services such as energy detection, measurement of the quality of a received packet, evaluation of the medium activity state, channel frequency selection and activation/deactivation of the radio transceiver occur [23].

2.2.2.2 MAC

The features of the MAC sub-layer are beacon management, frame validation, channel access and acknowledge frame delivery [22].

This layer is responsible for two different standard channel access methods: Beacon Enabled (BE) and a Non-Beacon Enabled (NBE) mode. NBE does not

send beacons nor superframes, in which MAC settles only on a non slotted CSMA/CA mechanism. On the other hand, BE sends periodically beacons to synchronize the nodes that are associated with, in order to identify the PAN.

Beacons contain the information on the addressing fields, the superframe specification, the Guaranteed Time Slots (GTSs) fields and pending address's [23]. The time between two consecutive beacon frames is referred as Beacon Interval (BI). When the standard selects this mode, the entire network is supported by a superframe. A beacon frame delimits the beginning of superframe, by defining a time interval during which frames are exchanged between different nodes in the PAN [21].

A Superframe is divided into an active period and an inactive period. The active period, referred as Superframe Duration (SD), is divided into the Contention Access Period (CAP) and the Contention Free Period (CFP). During the CAP, a slotted CSMA/CA algorithm is used for channel access, while in the CFP, communications occurs in a Time Division Multiple Access (TDMA) by using a wide number of GTSs [23]. When data transmission is not occurring (inactive period), the devices enters in the sleep state, in order to save energy [15].

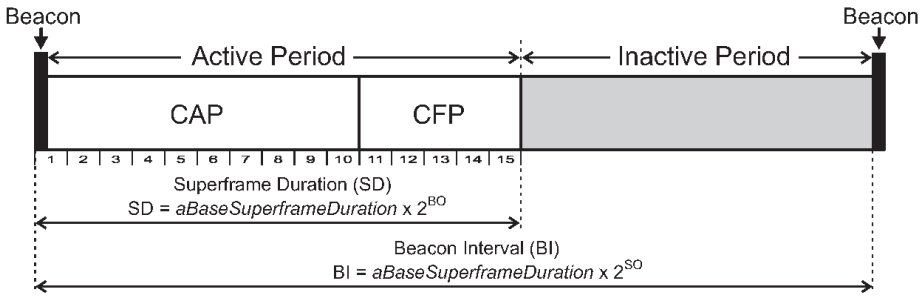

Figure 2.5: Superframe structure [1]

As depicted in Figure 2.5, BI and SD are determined by two parameters: the Beacon Order (BO) and the Superframe Order (SO) [25] [23], respectively. The "aBaseSuperFrameDuration" is the minimum superframe duration, which is equally divided into 16 time slots (0-15). The value of BO is associated with the maximum delay and packet arrival. of the sensor as shown in the following equation [26].

$$\delta = \frac{BI}{\lambda^{-1}} \tag{2.1}$$

where δ is the maximum delay and λ the arrival rate of packets from the sensor. SO influences the width of the active portion. The greater the SO, the wider component of the active portion.

The performance of the 802.15.4 MAC protocol, both in BE and NBE mode, presents a number of limitations and deficiencies. Since both modes relies on CSMA/CA algorithm, it cannot provide any bound on the maximum delay experienced by data to reach the final destination. Also, in BE mode, as a result of the same algorithm used for channel detection, the 802.15.4 standard provides a very low delivery ratio, regardless the number of nodes. Additionally, in BE mode, the use of multi-hop PAN with a tree topology requires the radio operation in a continuously way, which causes a large energy consumption [23].

Therefore, 802.15.4 is unsuitable for many critical scenarios, where applications such as ADAS have stringent requirements in terms of timeliness and reliability. However, it provides fundamental concepts useful for the next section.

2.3 MAC behaviors

As a solution, the IEEE 802.15.4e was proposed as an upgrade of the legacy IEEE 802.15.4 standard, to satisfy the requirements of emerging IoT applications [21]. The IEEE 802.15.4e defines five MAC behaviors, such as as Deterministic Synchronous Multichannel Extension (DSME), Time Slotted Channel Hopping (TSCH), Asynchronous Multi-Channel Adaptation (AMCA), Low Latency Deterministic Network (LLDN) and Radio Frequency Identification Blink (RFID).

The MAC behaviours introduce new crucial and remarkable enhancements, particularly for DSME, TSCH and LLDN. One of the most import enhancement is the multichannel access. The lack of multichannel access was one of the main disadvantage of the original protocol standard.

Additionally, beside the header and management layer based information elements used in IEEE 802.15.4, Information Elements (IE) have been introduced to support the five MAC behaviors. IE supports information regarding the number of superframes in a multi-superframe, number of channels, time synchronization specification, group acknowledgment and channel hopping specifications.

To fulfill the quality of service (QoS) required in IoT applications, the IEEE 802.14.4e implemented Enhanced Beacons (EB's). EB is a revision of the standard beacon format. They provide greater pliability and are used to supply application-specific beacon content to the DSME and TSCH MAC behaviors. Additionally, EB carries information on whether both MAC behaviors are enabled and information about the respective channel hopping sequences.

Moreover, the protocol supports a feedback to upper layers on the network performance. The feedback includes information regarding the number of frames that requires retries before acknowledgment (ACK), the ones that did and did not result in an ACK and the ones that were discarded due to security concerns [21].

The acknowledgments are supported in DSME and LLDN MAC behaviors as in successful missions can be acknowledge using a Group Acknowledgment (GACK) incorporated in the BI or as separated GACK frames.

The IEEE 802.15.4e protocol also provides an improved support for low latency communications, which is a main requirement for time critical applications. To fulfill the requirement, the standard specifies two mechanisms based on the latency requirements of the applications: Coordinates Sampled Listening (CSL) and Receiver Initiated Transmissions (RIT).

In CSL-enabled, there is a periodically channel verification for transmissions at low duty cycles. To reduce the transmitting overhead, coordination between transmitting and receiving devices are mandatory. For applications that run on low duty cycles and low traffic load, IEEE 802.15.4e supports the RIT mode.

In this thesis, we are only going to attend DSME, LLDN and TSCH since they are the three MAC behaviors that have channel diversity mechanisms, which enables the protocol implementation in networks within time critical applications.

We choose the DSME for our ADAS scenario since is the most suitable MAC behavior for similar applications, as described below.

2.3.1 Low latency Deterministic Network (LLDN)

The Low Latency Deterministic Network (LLDN) [21] is a MAC behavior suitable for applications that typically demand robustness due to the critical nature of the data. LLDN main target are network applications with a wide number of device nodes and a centralized control.

Usually, process control applications have a very small round-trip time and the communication has to be carried out in a periodic basis. To fulfill this requirement, LLDN's provide techniques for more determinism in small round-trip communications.

2.3.1.1 General aspects

LLDN's provide a exclusive star topology, in which all the nodes are individually connected to a PAN coordinator. Each node can either send data to the coordinator using an uplink or can send and receive data from it by using bidirectional time-slots.

Usually, uplink communications are used by sensors to transmit data related to physical magnitudes, while downlink communications are associated with actuators, which normally are monitored by a sensor.

Similar to DSME, the LLDN PAN coordinator uses superframes to transfer data.

2.3.1.2 Superframe

The LLDN PAN coordinator uses low latency superframes in order to transfer data. Each superframe is composed with a beacon, uplink timeslots, management time slots and bidirectional timeslots.

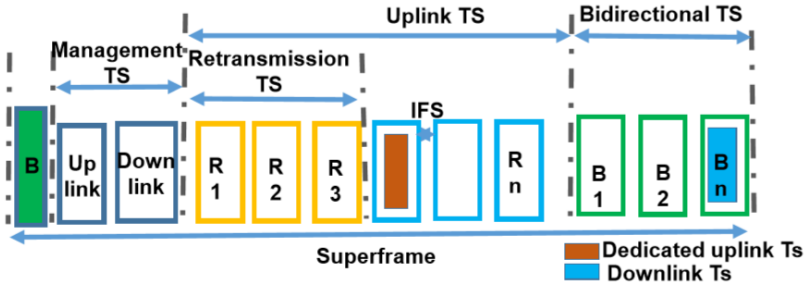

Figure 2.6: LLDN superframe structure [3]

As shown in Figure 2.6, the beacon occupies the start of every superframe and provides time synchronization for the entire network. Next in the superframe, downlink and uplink managment slots uses slotted CSMA/CA mechanism for channel access to send the management information. Moving forward, uplink timeslots are reserved for dedicated nodes assigned by the PAN coordinator, which only transmits in a unidirectional way. At least, the bidirectional timeslots are used for the communication from the PAN coordinator to the nodes. The direction of these slots are sent through an EB.

2.3.1.3 Transmission States

The network transmission begins with a Discovery state. During this phase, the superframe is composed of a beacon and two management timeslots. The association process in the LLDN starts by the orphan node, that will scan for a discovery EB, which was sent by the PAN coordinator [5]. After receiving the EB, the device uses an uplink management timeslot based on a CSMA/CA access channel mechanism to send a join request to the PAN coordinator. As a response, the PAN coordinator sends an ACK for each received data, including status frames.

Shortly after, the PAN coordinator switch its state to Configuration. During this phase, when a device receives a beacon indication the configuration state, it will send a Configuration Status frame to the LLDN PAN coordinator. The frame contains device configurations such as MAC address, timeslot duration required, uplink and downlink data communications and the assigned timeslots. The PAN coordinator, to respond to the configuration frames, sends a Configuration Request frame, which contains the new device configurations.

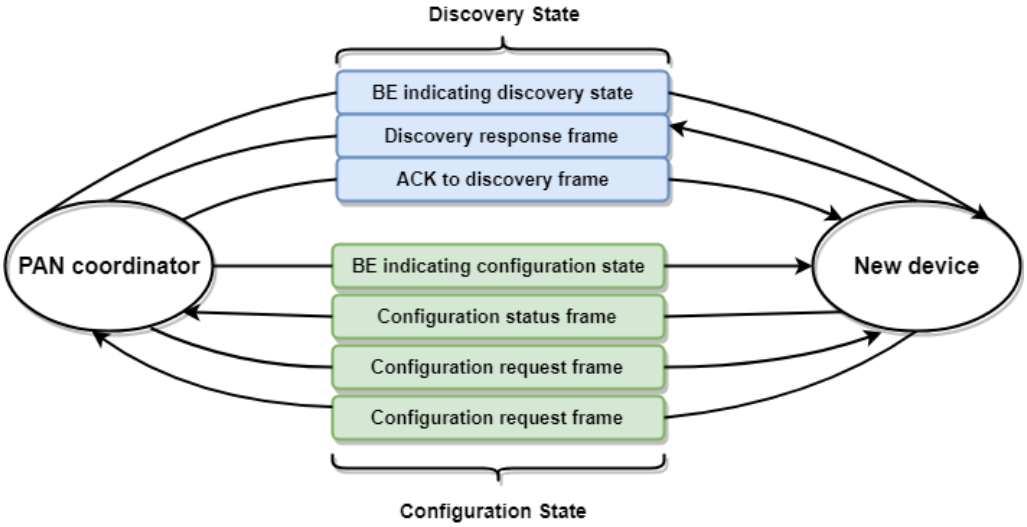

Figure 2.7: Discovery and Configuration state

After the introduced node sent an ACK confirming the Configuration Request reception, a new state begins. In this new online state, the superframe configuration is changed to support a beacon and several timeslots, based on the respective configuration. Almost identical to the DSME, LLDN provides GACK, which can be used in the uplink timeslots to ACK several re-transmission frames. When the node configures successfully, the device can send readings during the allocated slots through uplink communications with the online superframes.

2.3.1.4 Data Transmission Mechanisms

Transmissions can occur in two different ways: from a device to a PAN coordinator or from the PAN coordinator to a device.

When a device wants to send data to a PAN coordinator, it will wait until the reception of a beacon. After the reception, the device synchronizes according to the configuration received. To send data, the device can either use a dedicated timeslot, or a shared timeslot (shared group timeslots), using slotted CSMA/CA for the last case.

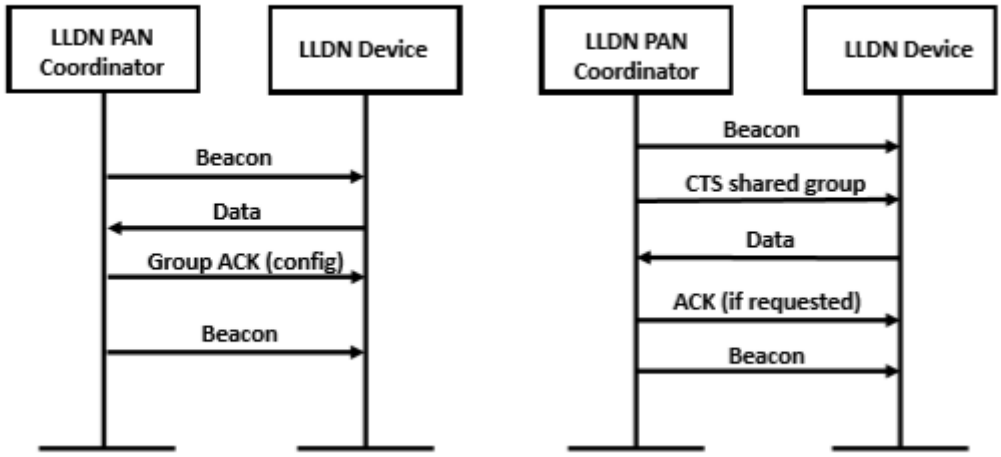

Figure 2.8: Communication from the LLDN PAN Coordinator to a device [4]

On the other hand, if the LLDN PAN coordinator wants to send data to a device, a bidirectional timeslot is assigned for transmission (either uplink or downlink). The PAN coordinator can configure the bidirectional timeslots to downlink and sends data frames without contention. Later, it waits for an ACK from the device upon a successful data reception.

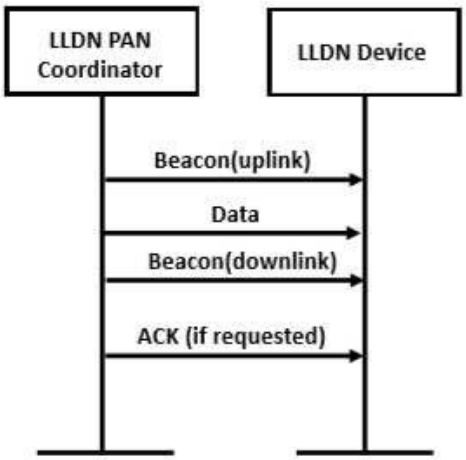

Figure 2.9: Communication from a device to the LLDN PAN Coordinator [4]

2.3.2 Time Slotted Channel Hopping (TSCH)

The Time Slotted Channel Hopping (TSCH) MAC behavior is a suitable candidate for implementing sensor-actuator networks in safety critical environments, which the prime concern are related to human and environmental safety. Applications such as oil and gas industries are prone to interfere and affect, in a direct way, the functionality of wireless devices. Fortunately, the use of TSCH as MAC behavior surpasses this issue.

2.3.2.1 General aspects

TSCH [27] [21] combines time-slotted MAC access, multi-channel communication and channel hopping. It divides time into parts of fixed length that are grouped in a frame. Time slots should be long enough to deliver data packages and to wait for an ACK between a pair of devices. Devices are synchronized and share the notion that each frame repeats over time, in a cyclic way. Time-slotted communications reduces the unwanted collisions that can lead to catastrophic failures.

Moreover, TSCH supports the channel hopping mechanism, which improves the reliability of the network by reducing the effects of interference at a considerable scale. Channel hopping is achieved by sending successive packages on different frequencies. TSCH uses the channel hopping sequences, which are fixed and known by all the nodes in the network.

Contrary to the previous MAC behaviors, the concept of superframe was replaced by a new one: slotframe.

2.3.2.2 Slotframes

Slotframes are a collection of different timeslots. All the timeslots accommodate a transmission and an eventual ACK, which could use CSMA/CA as contention, or be non contention based.

Slotframes repeats in cyclic periods, forming, therefore, a communication schedule. For identification, a slotframe handle is associated at the start of every slotframe.

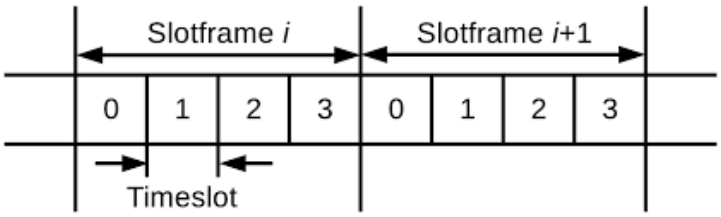

Figure 2.10: TSCH multiple slotframe structure [5]

Similar to DSME, concurrent slotframes can support concurrent transmissions, which are referred as multiple slotframes. Multiple slotframes should be aligned in order to configure different communication schedules and connectivity matrices to work in parallel.

2.3.2.3 Channel Hopping

The multichannel communication of TSCH completely relies on Channel Hopping. At the beginning, there are 16 available channels for communication, which are singularly defined by their ChannelOffSet. A link between the communication of "n" nodes is defined by a pair of values that specifies the timeslot in the slotframe and the channel offset used by the nodes during thenecessary transmission time. The frequency channel used for communication in a "n" timeslot composing the slotframe can be represented as

$$f = F[(ASN + ChannelOffset)\%Slength] \tag{2.2}$$

where "%" is the modulo operator, ASN is the timeslots elapsed since the start of the network and Slength is the length of the sequence, which may be larger than the 16 channels since some channels appear multiple times in the table "F", that contains a sequence of available physical channels.

The ASN works as a global timeslot counter, which is increased as new devices enter in the network. Consequently, the Equation 2.2 returns the communication frequency (channel) that a different channel can be implemented over the same link at different timeslots, defining, therefore, the TSCH Channel Hopping mechanism.

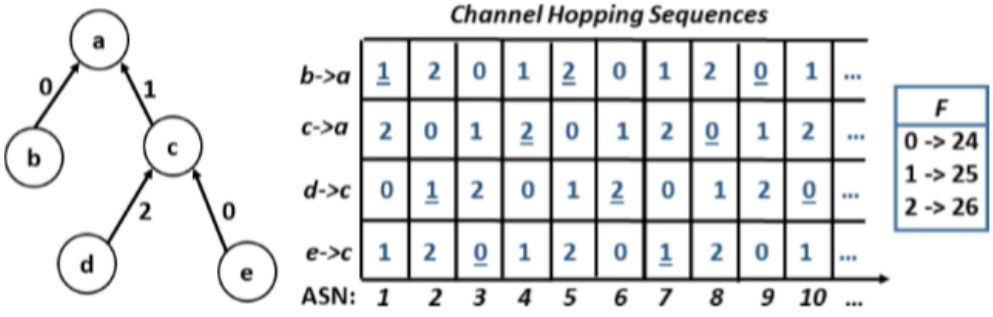

Figure 2.11: An example of Channel Hopping [5]

2.3.2.4 PAN Formation

With regard to increase the ASN, new devices should be in the broadcast range of the PAN coordinator to connect with it. The PAN coordinator broadcasts an EB which contains time information, Channel Hopping information, timelost information and the time on when to listen to an advertising device, by transmitting information to it (Initial Link Information), has a response to higher layers.

Devices wishing to join the network can either do an active or a passive scanning after receiving a SCAN request from a higher layer. Once the scan is done, the exact higher layer initializes the slot frame and the Initial Link Information, which is available in the EB, changing, therefore, the device into TSCH mode.

The size of the network plays a decisive role in determining the advertising rate and the configuration by the higher layer, which have a direct impact on the non functional properties such as scalability and power consumption of the network.

2.3.2.5 Time and Node Synchronization

Time synchronization outwards from the PAN coordinator in a TSCH based network. Devices must synchronize its network time with another neighboring device at periodic intervals in order to have a prior idea where the timeslot begins and where it ends.

Moreover, node-to-node synchronization is done to ensure the communication between near-by nodes in a slotframe based network. Time source devices keep track of neighbor devices and in the absence of receiving a request from them at least one per alive period, they will perform ACK based synchronization.

Synchronization could happen by two different methods: either through the time information that is received within an ACK from the receiver (ACK Based Method) or from the arrival time of a frame from the time source neighbor (Frame Based Synchronizations).

In ACK Based Method, synchronization is carried out through the exchange of data frames and ACK's. The receiver device estimates the error associated with the expected arrive time and the actual arrival time, which is sent to the transmitting node through an ACK.

On the other hand, in the Frame Based Synchronization, nodes synchronize to its own network clock, which are adjusted when the device receiver calculates the time difference between the expected arrival and the actual arrival time of a data frame from a time source neighbor.

2.3.2.6 CSMA/CA

The IE contains information regarding the decision to choose either TSCH mode or the slotted CSMA/CA mode in TSCH, which is issued during the network formation. If the link is established under the slotted CSMA/CA, it performs a Clear Channel Assessment (CCA), in which verifies the availability of the channel.

Similar to LLDN, TSCH links can be established either by shared links or via dedicated links.

Shared links are used to assign more than one device during data transmission. All the packet transmissions during this method should get an ACK as response.

Due to its wide number of transmission devices, these links are prone to collisions and failures. Therefore, re-transmission backoff algorithms are used. Every time this kind of algorithm is used, the back off window increases, reducing to the minimal value at a successful transmission.

The transmission in dedicated links is more reliable since there is no contention between devices to occupy the channel. The backoff window does not change when the transmission is successful, but increases as transmissions are added to the queue. When the queue becomes empty, the back off window is reset.

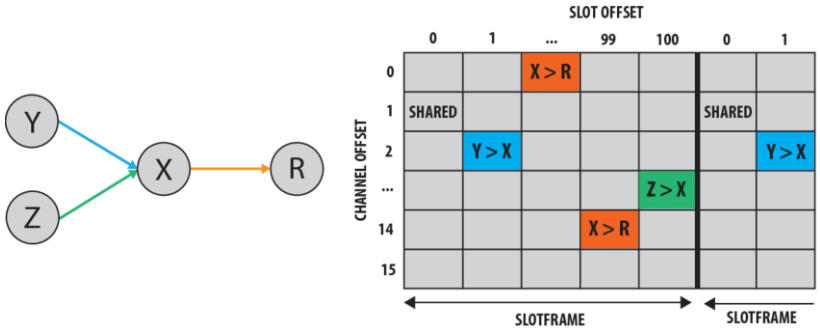

Figure 2.12: TSCH links establishment example [5]

2.3.3 Deterministic Synchronous Multichannel Extension (DSME)

The DSME critical MAC behavior aims to support industrial and commercial applications with stringent requirements in terms of timeliness and reliability [21]. Applications such as ADAS where latency, loss of data and information exchange are crucial factors can settle on this MAC behavior.

DSME derives from the BE mode defined in the former standard. In similarity with BE, DSME is divided into CAP and CFP. Although, the number of GTS timeslots is increased with the number of frequency channels used.

Near by nodes communicate with each other by dedicated links between any two nodes of the network. Therefore, DSME is an "ideal solution for covering multi-hop mesh networks with deterministic latency" [23].

2.3.3.1 Superframe

Contrary to the older standard protocol, in the DSME there are no inactive periods. As a result, the BI is full filled with different active superframes. The addiction of multiple superframes is referred as multi-superframes.

The MAC multi superframe Order (MO), which represents the beacon interval of a multi superframe, also has impact on the network performance. The multi super frame duration (MD) is obtained by the the following equation:

$$MD = aBaseSuperframeDuration * 2^{MO} \tag{2.3}$$

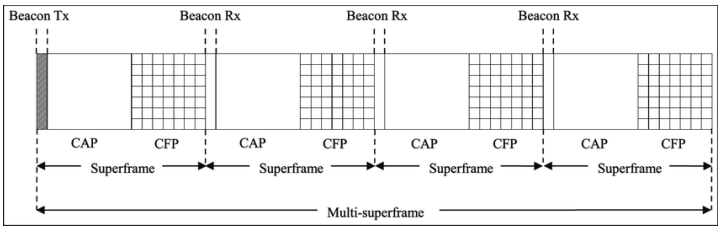

Figure 2.13: DSME Multi-Superframe structure [3]

The number of data packets transmitted synchronize the number of superframes in the multi-superframe, which are managed by the PAN coordinator. Each superframe is divided into 16 equally spaced slots (0-15). The slot number 0 is used to transmit EB's.

Immediately after the EB transmission, the CAP starts and ends before slot number 9 [19]. During this period, nodes uses CSMA/CA for medium access, in which all nodes are require to be ON, allowing, therefore, communications between any pair of nodes using CAP.

Slots 7 to 15 compose the CFP. Each slot takes advantage of the traditional GTS, by using a DSME Guaranteed Time Slot (DSME-GTS). Contrary to the CAP, in CFP there is no use of CSMA/CA. Therefore, CFP is usually used to transmit traffic and data frames with a predictable latency.

One of the disadvantages of the multi-superframes is the node activation during the CAP , which consumes valuable energy. As a solution, DSME MAC behavior provides the CAP Reduction Mechanism and the GACK.

2.3.3.2 CAP Reduction Mechanism

CAP Reduction Mechanism is "very a suitable add-on on highly dense networks with stringent QoS requirements in terms of delay and reliability" [21]. The reduction mechanism consists in only enabling the first superframe of each multi-superframe. As a consequence, the CAP is omitted in the other superframes, in which results in a longer CFP. The non-presence of the CAP allows nodes to be disabled when not transmitting, reducing, therefore, energy consumption.

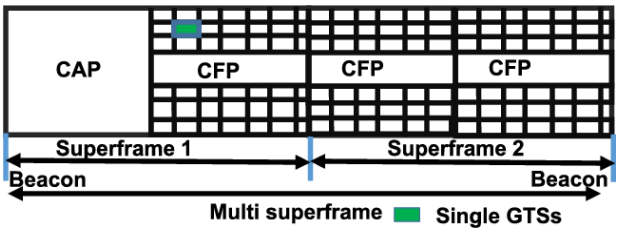

Figure 2.14: CAP Reduction Mechanism [3]

2.3.3.3 GACK

Group Acknowledgment improves energy efficiency by not acknowledging each received frame. Additionally, it reduces latency by provinding the opportunity of re-transmitting in the same multi-superframe [23].

GACK allows the coordinator to allocate two DSME-GTSs (GACK1 and GACK2), that are going be used to send an ACK in just one frame to the associated nodes. Each GACK acknowledges separated ID slot regions and are transmitted with the EB's that are sent by the coordinator. As a response, the receiver nodes indicates the reception status of each GTS and provide new slot allocations for failed transmissions or re-transmissions [21].

To guarantee the efficiency of the GACK, the transmission of the EB's must be trustworthy. To guarantee the reliability, the DSME provides a beacon scheduling mechanism.

2.3.3.4 Beacon Scheduling

To synchronize all the devices in a DSME network, the transmitted beacons from device to device contain the "Timestamp" field, allowing nodes to synchronize with neighboring nodes. As a node is introduced to the network, near by nodes transmit their beacon schedule on an EB. The introduced node searches for a beacon slot, and if available, will claim it to use it for sending its own beacons.

In the CAP of DSME, there is a risk of beacon collision as two or more nodes are trying to get the same beacon slot number due to neighboring transmissions.

As a solution, DSME presents a command, referred as DSME-Beacon Allocation Notification. The command divides the CAP into Allocation Contention Periods (ACP) and Permission Notification Periods (PNP). During ACP, when a node tries to allocate a not available slot, it transmits allocation notifications to coordinators, while during PNPs, only the entitled coordinator can transmit a permission notification. A node that has sent an allocation notification must wait for an explicit permission notification (sent by the coordinator that sent the latest beacon) in order to complete the allocation.

In case of two devices sending a allocation notification with the same STD Index (allocation superframe duration), the coordinator sends a collision notification in which decides which one of the nodes has a higher priority. The priority of each device is directly connected to the MAC performance metrics.

RFDs usually presents limitations regarding interference and transmissions failure. Therefore, DSME presents a mechanism of channel diversity to resolve the issue.

2.3.3.5 Channel Diversity

DSME networks allows neighboring nodes to communicate with other using any of the free available channels. The decision of the selected channel occurs during the DSME-GTS allocation phase, depending on the link quality between the two nodes. If the link quality decreases, the DSME-GTS is replaced with a more suitable DSME-GTS. This mechanism is referred as Channel Adaptation.

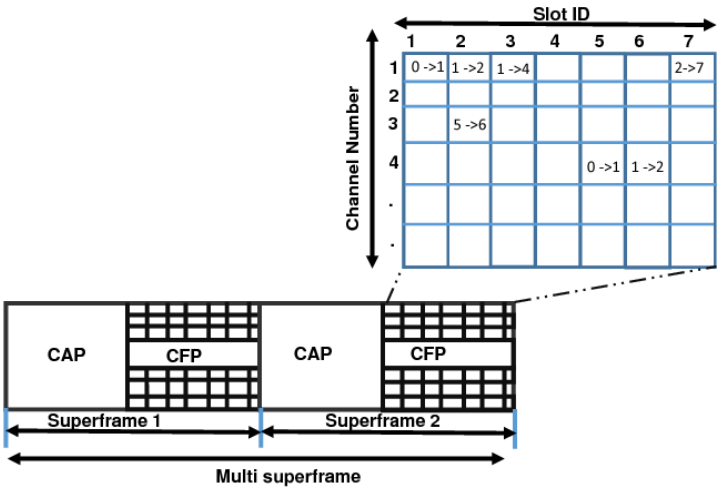

Figure 2.15: Channel adaptation example [3]

Additionally, nodes can communicate with each other by following a hopping sequence. In Hopping Sequence, the PAN coordinator decides the sequence that must be followed by all the nodes. The first position is referred as Channel Offset

0, and, in order to to establish communication between the following devices, the transmitting nodes have to switch to the channel used by the receiver.

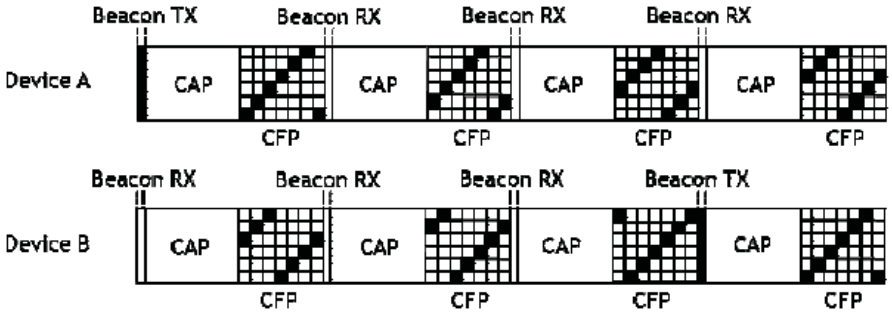

Figure 2.16: Hopping Sequence example

Chapter 3

Technologies and Tools

This chapter provides an overview of the technologies used throughout this thesis. Firstly, this chapter describes the frameworks, along with several necessary tools, used for the ADAS simulation scenario, including an overview of the network and robotics simulator.

3.1 Outline

When designing an intra-vehicle network, several preliminary steps should be considered. First, it is necessary to decide the use case in which we intend to implement our intra-car communication. Next, we need to choose the right technology to support it and the type of sensors that are intended to be used for our specific use case.

However, vehicle simulation tool (such as robotic simulators) although capable of simulating physics and control aspects of the vehicle, are not enough to simulate real life situations where communications support the underlying application/control aspects. Even though the communication between devices in complex systems are reliable, they are not guaranteed. Thus, the incorporation of a network simulator is necessary to obtain acceptable results.

Consequently, this chapter describes the fundamental concepts to fully understand the systems applied throughout this thesis.

3.2 Robot Operating System

Embedded systems such as ADAS require fast and scalable communications, cooperation of heterogeneous systems and QoS guarantees. To fulfill these requirements, middlewares such as ROS are required in these critical applications.

3.2.1 Review

Although it may seem like it, ROS is not an OS, but an open-source, meta-operating system for robots. However, it provides services that are expected from an OS such as hardware abstraction, low-level device control, implementation of commonly used functionality, message-passing between processes and package management [28]. Additionally, it provides tools and libraries to write, build and run code across multiple machines.

The task of dealing with real-world variations in complex mechanisms is so hard that no single individual, laboratory or institution can build a complete system from scratch [2]. Due to it's popularity, research and commercial companies adopted ROS as the software architecture of choice. This led to a combined effort on enabling resources to exchange software and knowledge. As a result, ROS now has a community level of experienced developers and documentation regarding different robotic applications. This ecosystem now has thousands of users word-wide, working from smaller projects to large industrial automation systems.

Currently, ROS provides two upgraded versions: Melodic and Kinetic, which are the 12th and 10th official ROS release, respectively. Melodic is supported on Ubuntu Artful and Bionic, along with Debian Stretch, while Kinetic only supports Ubuntu Wily and Xenial [28].

A variety of other robot frameworks are also available. Software such as CARMEN [29], MOOS [3] and Player [30] are also a valid option regarding robotics purposes. However, ROS community stands out for the ability of providing all the parts of a robot software system that otherwise would have to be written. This allows to focus on the crucial parts without worrying about the ones that matter the least [2].

Similar to CARMEN, messages transmission are made in a centralized way. This means that every message stream details must go though a central server. Also, from a technological perspective, the ROS can be looked at as the evolution of the Player framework, which up until ROS was the best example of an open-source software for his purpose [31]. Thus, from overall terms, the ROS is more suitable for tasks.

Additionally, the ROS achieves philosophical goals that no other framework supports.

3.2.2 ROS1

When it comes to designing ROS based applications, the most typical approach is the implementation of several on-board computers connected via Ethernet. As depicted in the Figure 3.1, the network is bridged via Wireless Local Area Network (WLAN) to off-board machines that are running computer-intensive tasks.

Regardless the location of the central server, the number of message routes would be fully contained in the subnets, which results in an unnecessary traffic flow across the wireless connection.

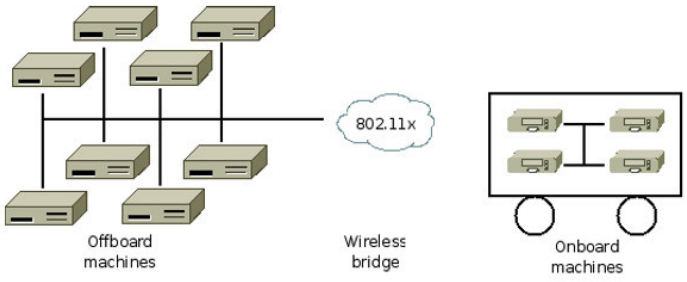

Figure 3.1: Typical ROS network configuration [6]

As a valid solution, Peer-to-Peer (P2P) systems are a popular approach when it comes to share such huge amounts of data [32]. Their reliability are due to the existence of links between two nodes. By knowing the location of each other, there is a directed edge from one node to the other one. Additionally, they can share fundamental resources between them. The ability of building an extremely efficient system by aggregating the resources of a large number of independent nodes enables P2P systems to increase the capabilities of many centralized systems such as the ROS.

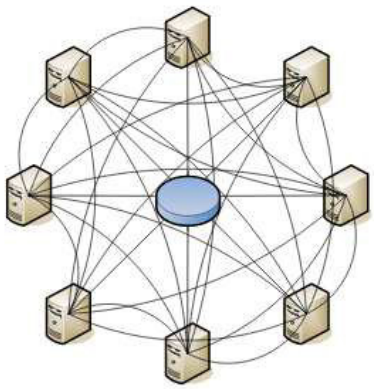

Figure 3.2: P2P centralized communication system [6]

Also, the ROS supports different programming languages such as Python, C++, Octave and LISP.

Each specification occurs at the messaging layer. P2P connections occurs in the central server, where implementations exist in most programming languages. To support this multi-lingual development, ROS uses a language-neutral Interface Definition Language (IDL) to describe the messages sent between modules. The IDL use text files to describe fields of each message. Before being transmitted over the network, code generators for each supported language create implementations that are automatically serialized and deserialized by the ROS, as messages are sent and received [6]. The message passing scheme allows the collaboration between several developers on robotic applications, regardless the language used.

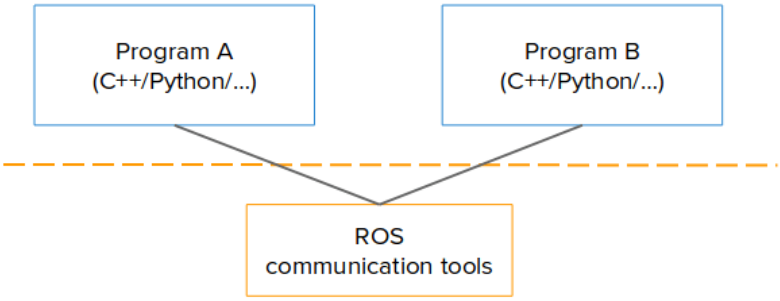

Figure 3.3: ROS communication tools scheme [6]

Unlike many other robotic software frameworks, the ROS takes advantage of small tools that are used to build and run the various ROS components, instead of a monolithic development and run-time environment [6].

Separating services into different modules allows the replacement of better-fitting implementations for a particular task domain. In terms of community, this is an asset when it comes to keep on track with technological leaps.

Moreover, the ROS encourage all driver and algorithm development to occur in standalone libraries that have no dependencies over the framework. This is possible due to the build system performing on modular builds inside the source code tree [6].

The idea of abstracting complex and general use code into different libraries allows the creation of smaller programs, in which translates into an easier way to develop and reuse software applications.

Naturally, achieving these philosophical goals allowed the ROS to emerge as one of the most used robotics framework. To ensure that all the users can take full potential of the platform, the ROS settles on fundamental communication

concepts that allows most recent developers to full understand the systems and concepts that are required to design robotic software applications.

The ROS communication architecture can be seen as a simple system that includes five keywords: services, nodes, topics, publications and subscriptions.

Basically, in a general way, a node is a program that communicates with another node through a specific topic. The communication system settles on the exchange of publications and subscriptions of data from a a large variety of nodes. Different nodes can publish or/and subscribe to a heterogeneity of different topics of different applications. On the other hand, nodes can invoke services that can only be provided by a single service node.

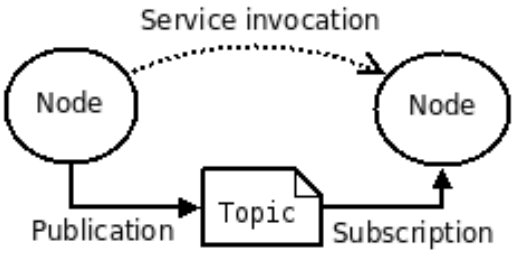

Figure 3.4: ROS basic communication concepts [6]

As mentioned before, the ROS relies on a centralized communication system. The central server, or "Master", is implemented via XMLRPC, which is a stateless HTTP-based protocol. It has registration Application Programming Interfaces (APIs) which permits programs from different languages to register as publishers, subscribers and service providers.

The central server is not used for data transport. Instead, it only negotiates connections with other nodes in order to establish communications between them.

Regarding off-board machines, the master has a Uniform Resource Identifier (URL) which corresponds to the port of the host that supports the server, allowing, therefore, the node connection.

As part of the Master, the parameter server, which is also implemented via XMLRPC, is a shared dictionary that is accessible via network APIs. The ROS parameters have a hierarchy that matches the name spaces used for topics and nodes, allowing, therefore, to protect parameter names from colliding.

Nodes have a slave XMLRPC API, which receives callbacks from the Master to negotiate connections with others. These callbacks updates contain a topic name and a list of URLs of nodes that publish that topic [33]. Moreover, the XMLRPC server receives calls from subscribers that are searching to request topic connections. To identify the source, each node also has a URL that corresponds to the port of the host that runs the XMLRPC server.

Services can be looked as a simplified version of topics, where the most recent node to register with the master is considered the current service provider, by exchanging a connection header.

A node subscriber is informed by the Master of the new publisher. Later, the subscriber contacts the publisher to request a topic connection. As a response, the publisher selects a supported protocol and returns the necessary settings to establish the data exchange, such as the IP address or the port of a TCP/IP server socket.

Naturally, nodes support topic transport protocol implementations. Topic transports can settle on both TCP and UDP, if the connection is established via Ethernet or Wi-Fi, respectively.

Figure 3.5 represents an example of two nodes, in which the ”Listener” subscribes the data published by the ”Talker”.

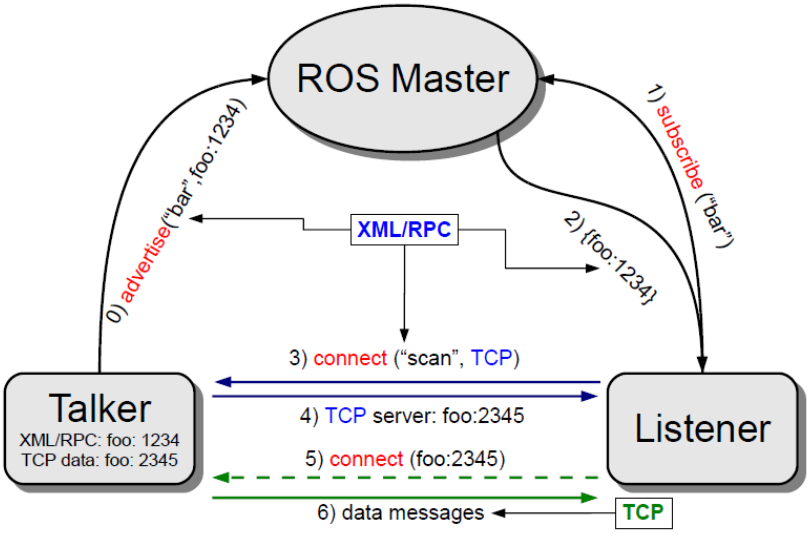

Figure 3.5: Communication architecture of two nodes [6]

3.2.3 ROS 2

As mentioned above, the ROS is not suitable for real-time requirements, can not guarantee fault-tolerance, deadlines, or process synchronization [18]. For that reason, Advanced Driver-Assistance Systems (ADAS) can not use the ROS as middleware since their applications demands high reliability regarding time critical factors.

To overcome this restriction, the ROS community launched a significant upgrade that is able to fulfill new use cases. The new version of this framework is referred as ROS2.

ROS2 has only one available version: Crystal, which was released in December 2018. Crystal supports Ubuntu 18.04 (Bionic), MAC OX 10.12 and Windows 10.

This new version of the ROS aims to be suitable for new use cases such as real-time systems, small embedded platforms, non-ideal networks and cross-platforms. In order to fulfill these requirements, the previous ROS (ROS1) transportation system was replaced by a new technology that provides a reliable publish/subscribe transport similar to that of ROS1, namely Data Distribution Service (DDS).

DDS allows to operate applications in a wide variety of Operating Systems (OS), namely Real-Time Operating Systems (RTOS). RTOS supports real-time systems, which are time bounded systems with fixed time constraints. This allows the incorporation of DDS on IoT applications.

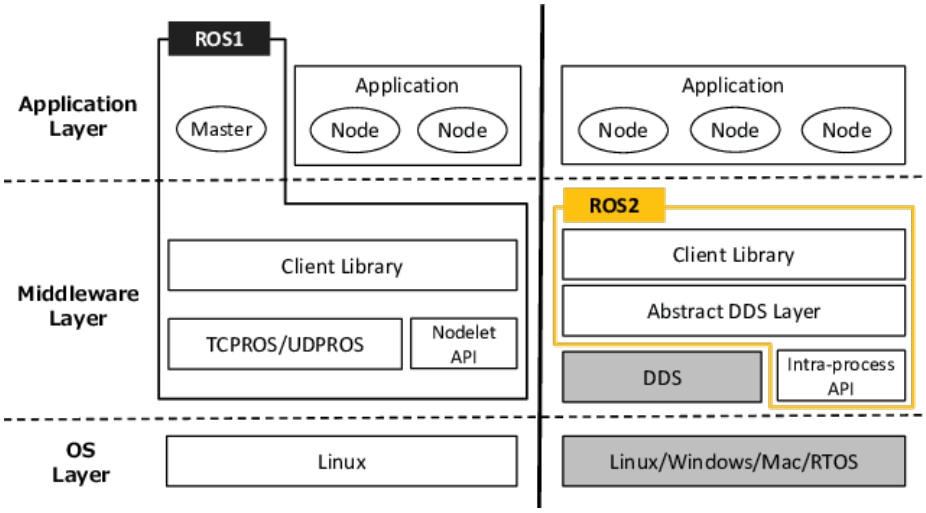

Figure 3.6: ROS1 vs ROS2 layer architecture [6]

Additionally, the ROS1 was reconstructed to improve user-interface APIs and incorporate similar technologies to complement DDS, such as Zeroconf, Protocol Buffers, ZeroMQ, Redis and WebSockets [34]. In this thesis, we are only going to refer DDS as it is the main technology that enables the new use cases.

DDS is one of the many protocols used in industry sectors. Applications such as air traffic control, medical services, military and aerospace takes advantage of the wide set of QoS parameters including durability, presentation, lifespan, reliability and deadlines that DDS provides [35].

DDS is standardized by the Object Management Group (OMG) and provides a Data-Centric Publish-Subscribe (DCPS) middleware for dynamic distributed systems. The DCPS model supports a Global Data Space (GDS) that can be accessed by any participant. Each participant can read and write from the GDS using a typed interface.

Similar to ROS, an application that sends data to one or more topics is referred as a Publisher. A Publisher takes use of a DataWriter, which is a object that must be used by a participant to publish data of a certain type. Accordingly, a Subscriber application that is responsible for receiving published data uses DataReader, which can receive and access data whose type must correspond to that of the DataWriter.

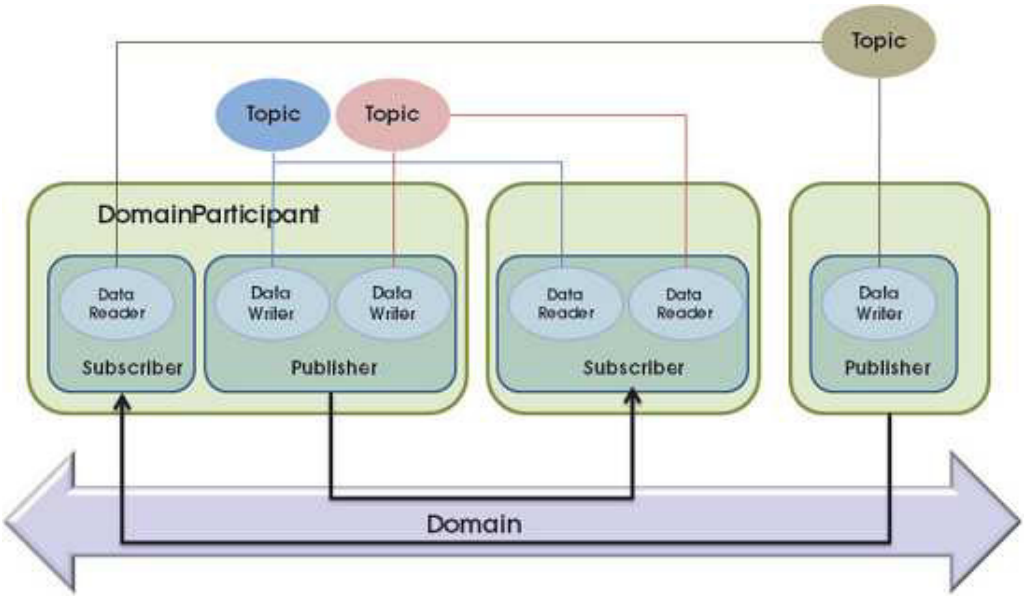

Figure 3.7: DDS transport system representation [7]

To fulfill the real-time requirements, all the DSPS entities have a QoS Policy, which represents their data transport behavior. The information could be related to deadline period, depth of history or even communication reliability. In DDS, there are a wide variety of other QoS Policies [36], which ROS2 supports in order to extend its capabilities [18].

Data is published into the DCPS and the subscriber nodes can get the data without knowing the source of the information or how it is structured, since the information package already describes itself [35].

After a DataReader identify a topic name to subscribe, a DataWrite connects to a DataReader using the Real-Time Publish-Subscribe (RTPS) protocol in distributed systems. The connection of the DDS standard with the RTPS protocol allows different DDS implementations to inter-operate by abstracting and improving data transport, namely TCP, UDP and IP.

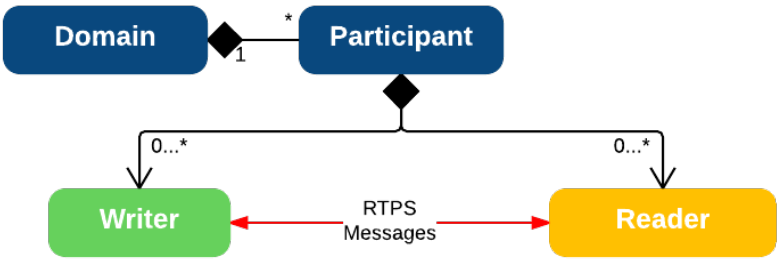

Figure 3.8: DataReader and DataWriter message exchanging [6]

The introduction of the DCPS into the ROS architecture brought the possibility of managing the node communication system without the requirement of a central server. Moreover, using UDP as its transport, DDS gives control over the level of reliability a device can expect and act accordingly, which is a crucial aspect regarding WSNs. Therefore, ROS2 is introduced to the robotics community as a valid middleware option regarding the IoT paradigm.

3.3 Gazebo

Robot simulation is an essential tool in every robotic design. A well-designed simulator makes it possible to test algorithms, perform regression testing and monitoring the performance of sensors and actuators data.

With this in mind, Gazebo offers the ability to accurately and efficiently simulate robotic systems in complex indoor and outdoor environments [37].

Gazebo [38] is a ROS-based robotic simulator which has a Client/Server architecture and a Puslish/Subscribe model of inter-process communication. The clients can access data through a shared memory. Simulation objects, displayed in advanced 3D graphics, can be associated with one or more controllers that process commands to control the object in a dynamic simulation with multiple high-performance physics engines.

The controllers create data that are published into the shared memory using the Gazebo interfaces. The interfaces of other processes can read the data from the shared memory, allowing, therefore, inter-process communication between the robot control software and Gazebo.

Likewise, Gazebo uses ROS features in many ways to achieve proper simulation synchronization and communication between different software modules on simulation scenarios [39].

Figure 3.9 shows a robot sheep named Dolly. Dolly [8] follows a person while carrying heavy stuff. It has motorized wheels that allow the robot go guide itself with a laser scanner to detect objects ahead.

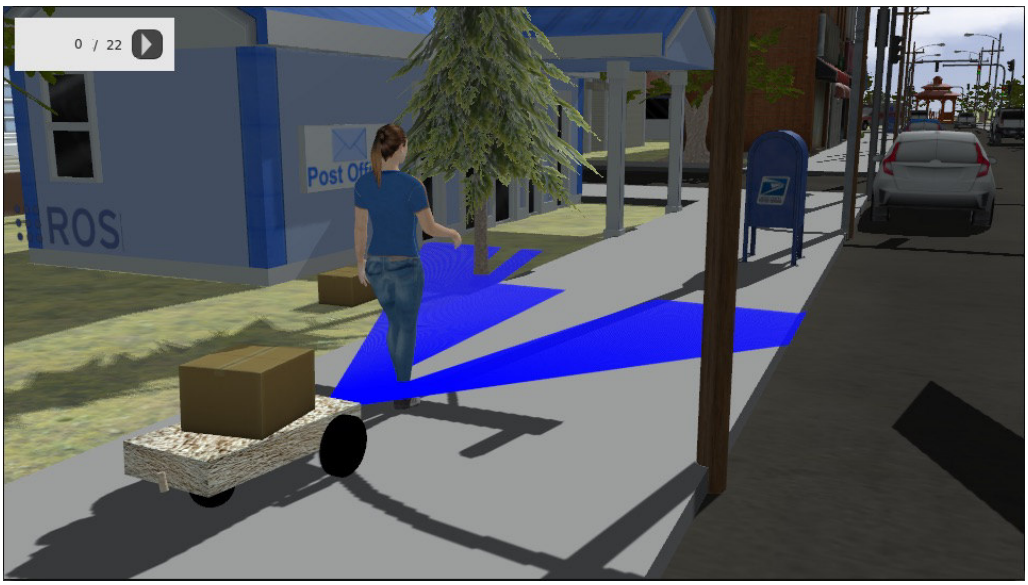

Figure 3.9: Gazebo representation of Dolly [8]

The clients send control data and objects coordinates to the server, which performs real-time control of the robot application. As a response, the server send sensor and position data to the client.

Regarding sensors and actuators, Gazebo supplies plugins for robots/environment control, developed with Gazebo's extensive API. Each information and control are set through a command line tools which runs under ROS.

3.4 Rviz

Rviz (ROS visualization) is a 3D visualizer for displaying sensor data and state information from ROS [40]. This tool allows to motorize what the robot is "seeing", "thinking" and "doing".

There are two main ways of putting data inside the Rviz world. First, Rviz understands sensor and state information like laser scans, point clouds, cameras and coordinate frames. These have specialized displays that allows to configure how the user wants to view information. Moreover, Rviz has visualization markers that allows the programmer to send primitives like cubes, arrows and lines colored however the user wants. The combination of sensor data and custom visualization markers make Rviz a powerful tool for development of robot capabilities and research.

Figure 3.10 illustrates a Rviz plugin of world famous robot, Nao. The plugin allows us to control and observe all movements of the robot, and monitor all its surrounding sensors.

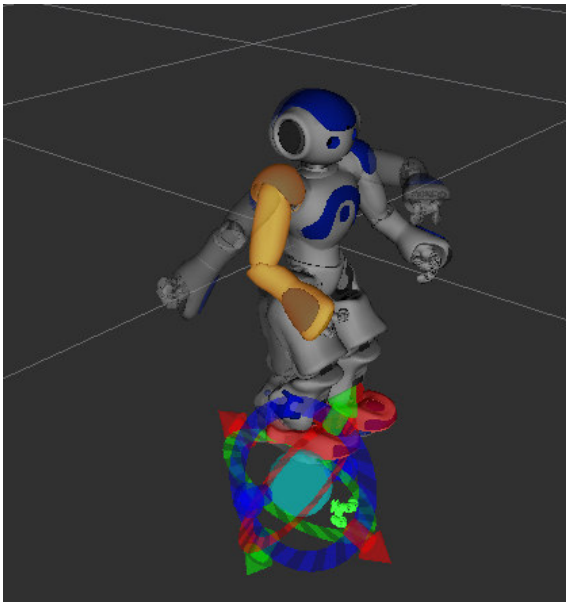

Figure 3.10: Nao robot plugin

The ROS navigation stack uses a combination of data to show its current path, obstacle data and topological map. Moreover, markers have also been used by motion planning researchers to show planned versus actual paths, object detection and calibration.

3.5 OMNeT++

Network simulation frameworks are important tools to evaluate new approaches in large scale scenarios with many real world environment networked nodes.

Several simulation frameworks have been widely used in networking research scenarios, including [41] ns-2, GoMoSim, OPNET, SensorSim, J-Sim, OMNeT++, etc..

Although, further research indicated that OMNeT++ has better performance than ns-2 in terms of execution time and memory overhead [42], and has the richest animated Graphical User Interface (GUI), which makes OMNeT++ a prior simulator to implement real model's.

Objective Modular Network Testbed in C++ (OMNeT++) [43] is an object-oriented modular discrete event simulator, mainly focused on the simulation of communication networks [41].

OMNeT++ aims to fill the gap between open-source, research-oriented simulation software and expensive commercial alternatives. Therefore, OMNeT++ is public-source, and can be used under the Academic Public Licence (APL) that makes the software free for non-profit use.

OMNeT++ was designed to support network simulation on a large scale. Consequently, simulation model's need to be hierarchical, and build from reusable component as much as possible.

In order to support a large scale, the simulation software itself is modular, customizable and allows embedding simulations into larger applications. Reusing modules allows users to not design network systems from scratch, focusing only in the message passing between the modules.

Although, different modules have different inputs and different outputs. These dependencies difficult the task of reusing modules. As a solution, OMNeT++ allows to generate and process input and output files with commonly available software tools. This feature allows the adaptation of different modules for specific applications.

Additionally, the simulation framework provides an IDE that largely facilitates model development and analyzing results. Traditionally, debugging time takes up a large percentage of simulation projects, in which OMNeT++ facilitates the visualization and debugging of simulation model's.

The active modules are referred as simple modules. Simple modules are written in C++ and can be grouped into compound modules and so forth. A module can be built within the connection of other modules (sub-modules). Messages passing can be either via connections that span between modules or directly to their destination modules.

Simple modules send message via gates, which are the input and output interface of modules. An input and an output gate can be linked with a connection. Connections are created within a single level of connectivity. To a compound module corresponds gates of two sub-modules. A gate of one sub-module and a gate of the compound module can be connected. Connections spanning across different types of modules are not allowed.

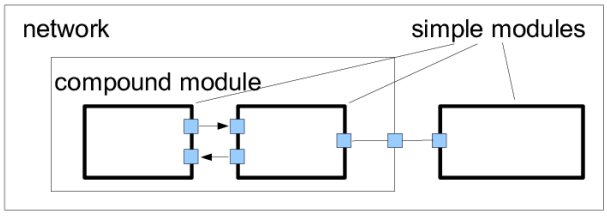

Figure 3.11: OMNeT++ module communication [6]

In most typical network simulations, messages travel through a chain of connections, to start and arrive in simple modules. Moreover, each module can have parameters. Parameters may be strings, numeric or boolean values and are used to pass configuration data to simple modules to define model topology.

To define the model structure, the user defines in OMNeT++'s topology description language, NED. NED consists on simple module declarations, compound module definitions and network definitions.

Module declarations describes the interface of the module (gate and parameters). Equally, compound module definitions consist on the declaration of the module's external interface and the interconnection and definitions of the submodules. At least, network definitions are compound modules that qualify as self-contained simulation model's.

Model behavior is captured in C++ files as code, while model topology is defined by the NED files. Naturally, OMNeT++ model's modules contain three types of files defining it, one c++ code application file, a c++ header and a .ned file describing the module [39].

3.6 openDSME

openDSME [44] [45] is an open source implementation of IEEE 802.15.4. DSME. It is portable for various platforms, mainly simulation environments and hardware platforms. Moreover, the implementation is a C++ data link layer that can run in the OMNeT++ simulation environment using the INET framework.

Instead of providing just another full-stack OS for WSNs, this implementation provides adaptation layers, which employs a simple traffic based slot reservation scheme, to plug it into existing stacks for a evaluation of the MAC behavior in existing hardware and software environments. Each adaptation layer can be combined with any network layer on top.

These same layers have already been created for OMNeT++/INET, and for other similar network simulators, such as cometOS [46].

Chapter 4

Co-Simulation framework for wireless ADAS

This chapter describes in great detail the design and implementation of the ADAS co-simulation scenario, dividing it into sections that focus in the cooperation between each different simulation framework.

4.1 Scenario Specification

For the ADAS scenario, we consider a use case where a classic sonar-enabled intra-car system is used to detect the surroundings for any threat and, consecutively, alerts the user and tries to prevent an accident.

These systems are not a new archetype regarding the automotive sector. Vehicles such as the Tesla model X exploits eight surround cameras that provides 360 degrees of visibility around the car, twelve updated ultrasonic sensors, and a forward-facing radar to detect other cars in the environment and move safely.

Figure 4.1: Tesla X advanced sensor coverage [9]

Our goal is not to upgrade the quality of these structures, but to design our own system in such a way that it can be implemented on any vehicle.

To test the use case, in this thesis, we propose the implementation of a scenario in which, Sound Navigation and Ranging (SONARs) are used to detect the surrounding environment of a vehicle model. The sonars in the intra-car model have to be strategically placed towards the corners of the car to achieve maximum field of view without any blind spots. The vehicle should be lined up with an obstacle, in which two different situations could occur: the vehicle overtake the obstacle and return to the previous direction, or, in case of other vehicle is overtaking the model, the car chooses to stop the movement.

4.2 System Architecture

To build our own simulation, we are going to use a demo of a Hybrid Prius as a example of vehicle model, which is available for the cooperative performance between ROS and Gazebo [10].

4.2.1 ROS-Gazebo Interface

ROS and Gazebo communicate between each other through a package (gazebo_ros), which provides ROS plugins that offer message and service publishers for interfacing with Gazebo.

The car demo follows real-life features, such as design, weight, dimensions, speed restrictions and steering angles, in order to approximate the vehicle control to the original real-life model.

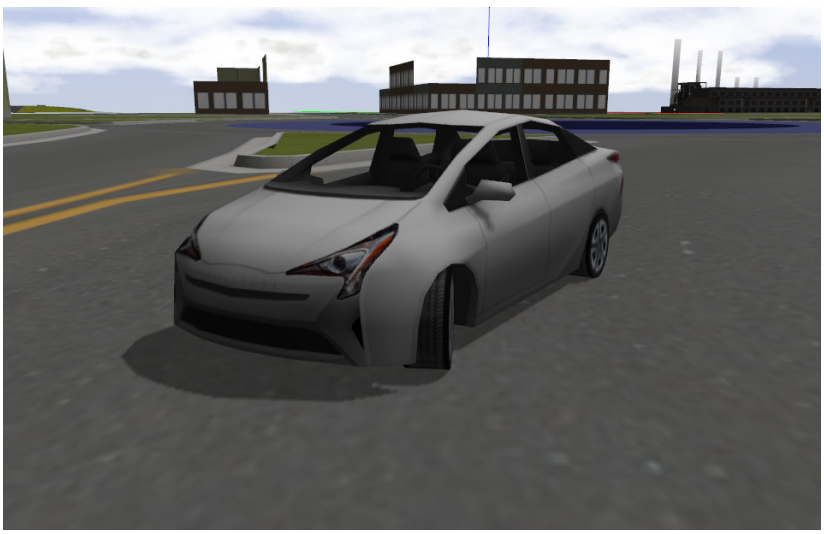

Figure 4.2: Hybrid Prius as a vehicle model [10]

The Hybrid Prius is described in a Unified Robot Description Format (URDF) [47] text file. URDF follow a XML format and are used to model a robot with Links connected by Joint's in a chain or tree. Multiple function robots can be modeled with a tree data structure of Joint's connected by Links to a base link.

Joint's provide relative motion between two Links of the robot and Links provide a certain degree-of-freedom of motion.

Links can contain elements for visual, inertial and collision properties. All of these elements are related to an origin, which works as the reference frame of the visual, inertial or collision elements with respect to the reference frame of the link.

Visual elements specifies the shape of the object (cylinder, box, sphere, etc.) and the type of the material for visualization purposes. Inertial elements are composed with the representation of the value attributed for the mass of the link and by the 3x3 rotational inertia matrix, which is essentially the rotational equivalent of mass since it relates the angular momentum of a system to its angular velocity. Finally, the collision elements are related to the physical component of a link [11].

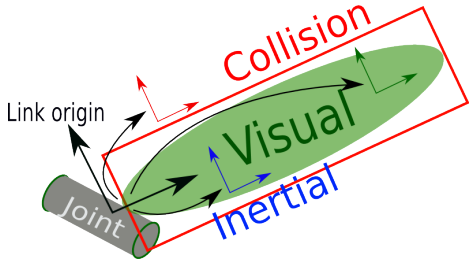

Figure 4.3: Link elements [11]

Each link is moved by a single joint. However, Links can connect to a wider number of Joint's that may move other Links.

Joint's contain elements for origin, child link name, parent link name, axis of rotation or translation, dynamics, calibration, limits and safety controller information [47].

Joint's must specifie their type, where it can be one of the following.

- Revolute: a joint that rotates along the axis and has a limited range specified by the upper and lower limits,
- Continuous: a joint that rotates around the axis and has no upper and lower limits,
- Prismatic: a joint that slides along the axis and has a limited range specified by the upper and lower limits,

- Floating: a joint that allows motion for all 6 degrees of freedom
- Planar: a joint that allows motion in a plane perpendicular to the axis

The joint attaches a parent link to a child link. In a chain or in a tree, the child link can be a parent link to one or more Links. The connection of a child link through a series of Joint's and Links back to its parent is not allowed, since there is no use in implementing a loop.

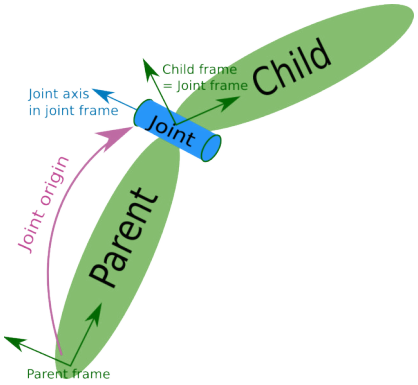

Figure 4.4: Joint elements [12]

The Hybrid Prius specification can be seen as a tree structure of rigid links connected by different types of joints. This fundamental knowledge is crucial to understand the sensor implementation.

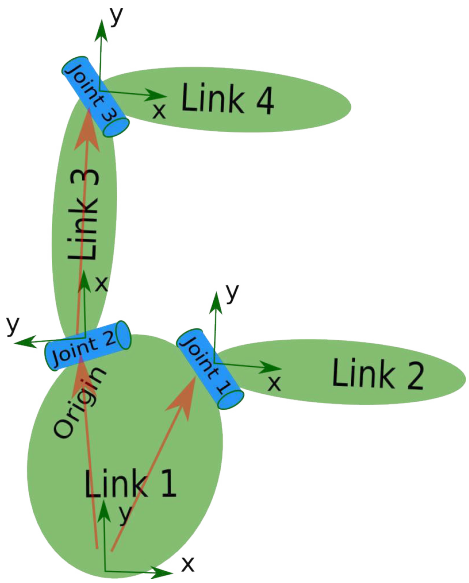

Figure 4.5: Analogy of the Hybrid Prius structure [12]

This simulation of a Prius in Gazebo works as a node, named as "PriusHybrid-Plugin". The model publishes sensor data using ROS, in which the framework can control the throttle, brake, steering and gear shifting, by publishing a ROS message into each specific constant of a topic referred as "prius".

Constants from different controllers assume different values. Throttle and breaking supports values between 0 to 1, in which 1 is the maximum throttle/brake, and 0 is the minimum one. Steering accepts values between -1 and 1, which translates into the maximum right turn and the maximum left turn, respectively. The gear control is not necessary for the simulation. Therefore, the vehicle will be controlled by the Powertrain model of the car.

Each "Control" communication is defined into a .MSG file, which are a simplified message description language for describing the data values that ROS publishes. .MSG files are separated into fields and constants. Fields are data that is sent inside of the message and constants define values that can be used to interpret those fields.

The vehicle model requires a physical world in order to move around. In this thesis, the environment used for this scenario was also developed by us, which we provide a .WORLD file describing a circuit. Both the vehicle model and the world are generated through .LAUNCH files, which uses the XML format to run similar files.

To fulfill the requirements for the two different situations explained before, the .LAUNCH file regarding the prius model includes two different demos, named as car1 and car2, respectively, in which the initial position of each vehicle is defined through a Cartesian axis.

To interact with the surrounded environment, the prius model simulate proximity sensors that publish data to a specific topic. In this scenario, we implemented SONARs that have very similar characteristics with a Devantech SRF08 High Performance Ultrasonic Range Finder, which is a right fit to detect the vehicles that try to overtake or move towards the side of the car. Additionally, this SONAR consumes significantly less power (15mA) and can detect up to 16 returning echos, which benefits high velocities and several obstacle detection within different distances inside the field of view.

We do not claim that the SRF08 is the best SONAR for ADAS. In fact, we do not recommend any specific SONAR for this kind of systems. However, SRF08 properties fit in the required specifications to test this use of case.

The specifications of each sensor are described into a Gazebo reference made for sensors. Figure 4.6 expose that each implemented sensor was set to cover a maximum range up to 9 meters, a horizontal angle of 0.30rad and a vertical angle of 0.14rad towards the corners of the vehicle. Moreover, the field of view is repre-

sented with multiple samples that publishes data to the topic car2/sensor/sonarX, where X represents the number of the SONAR at 5 Hz frequency.

To present a more realistic environment, we added Gaussian noise to the data generated by the each sensor. Gaussian noise is caused by natural sources such as vibration of atoms due to high temperature and the radiation of warm devices [48].

prius.urdf

```
<gazebo reference="base_sonar_direito2">
   <sensor type="ray" name="direito2_sonar">
      <pose>0 0 0 0 0 0</pose>
      <visualize>true</visualize>
      <update_rate>50</update_rate>
      <ray>
         <scan>
            <horizontal>
               <samples>10</samples>
               <resolution>1</resolution>
               <min_angle>-0.30</min_angle>
               <max_angle>0.30</max_angle>
            </horizontal>
            <vertical>
               <samples>10</samples>
               <resolution>1</resolution>
               <min_angle>-0.14</min_angle>
               <max_angle>0.14</max_angle>
            </vertical>
         </scan>
         <range>
            <min>0.1</min>
            <max>9</max>
            <resolution>0.02</resolution>
         </range>
      </ray>
      <plugin filename="libgazebo_ros_range.so" name="gazebo_ros_range">
         <gaussianNoise>0.005</gaussianNoise>
         <alwaysOn>true</alwaysOn>
         <updateRate>50</updateRate>
         <topicName>sensor/sonar2</topicName>
         <frameName>base_sonar_direito2</frameName>
         <!--radiation>INFRARED</radiation-->
         <fov>0.2967</fov>
      </plugin>
   </sensor>
   </gazebo>
```

Figure 4.6: .URDF file excerpt of the specifications of each SONAR

As mentioned above, each sensor must specify the link visual elements of the object and the Joint specification elements.

As depicted in Figure 4.7, each sensor was simulated as a box with $125cm^3$ and with a standard physical sensor characteristics, such as inertial and collision. All the sensors are going to behave as continuous joint's fixed in the chassis, which works as a parent link.

prius.urdf

```
631  <link name="base_sonar_direito2">
632      <collision>
633        <origin xyz="0 0 0" rpy="0 0 0"/>
634        <geometry>
635          <box size="0.01 0.01 0.01"/>
636        </geometry>
637      </collision>
638
639      <visual>
640        <origin xyz="0 0 0" rpy="0 0 0"/>
641        <geometry>
642          <box size="0.05 0.05 0.05"/>
643        </geometry>
644      </visual>
645
646      <inertial>
647        <mass value="1e-5" />
648        <origin xyz="0 0 0" rpy="0 0 0"/>
649        <inertia ixx="1e-6" ixy="0" ixz="0" iyy="1e-6" iyz="0" izz="1e-6" />
650      </inertial>
651    </link>
652
653    <joint name="sonar_direito2_joint" type="fixed">
654      <!--axis xyz="-0.75 -1.4 0.9" /-->
655      <origin rpy="0 -0.1 3.9" xyz="-0.9 -1.7 0.6" />
656      <parent link="chassis"/>
657      <child link="base_sonar_direito2"/>
658    </joint>
```

Figure 4.7: .URDF file excerpt of the links and joints of each SONAR

The sensor data communication is defined into a .MSG file, in which has constants related to the type of radiation, the field of view, the minimum and maximum range and the distance between the sensor and an obstacle.

Range.h

```
47    typedef  ::std_msgs::Header_<ContainerAllocator>  _header_type;
48   _header_type header;
49
50    typedef uint8_t _radiation_type_type;
51   _radiation_type_type radiation_type;
52
53    typedef float _field_of_view_type;
54   _field_of_view_type field_of_view;
55
56    typedef float _min_range_type;
57   _min_range_type min_range;
58
59    typedef float _max_range_type;
60   _max_range_type max_range;
61
62    typedef float _range_type;
63   _range_type range;
```

Figure 4.8: .MSG file of the sensor

At total, 8 SONARs were implemented around the car2 model to ensure the analyses of the network performance with multiple devices and to guarantee that the obstacle and the car1 overtake is detected by the sensors.

Figure 4.9: SONARs implementation over the chassis of the Prius

Complex tree structures require a monitoring system since many 3D coordinate frames change over time. Thus, ROS provides a ”tf” library, which was designed to provide a standard way to keep track of the coordinate frames and to produce individual data of the coordinate frame wanted, without requiring all the coordinate frames in the system [49]. Rviz provides tools capable of producing 3D representations of each coordinate frame.

The obtained sensing areas are graphically described as markers in Rviz. The current distances within these areas to any object present in the simulation is presented in Rviz, which allows to monitor the behavior of each SONAR.

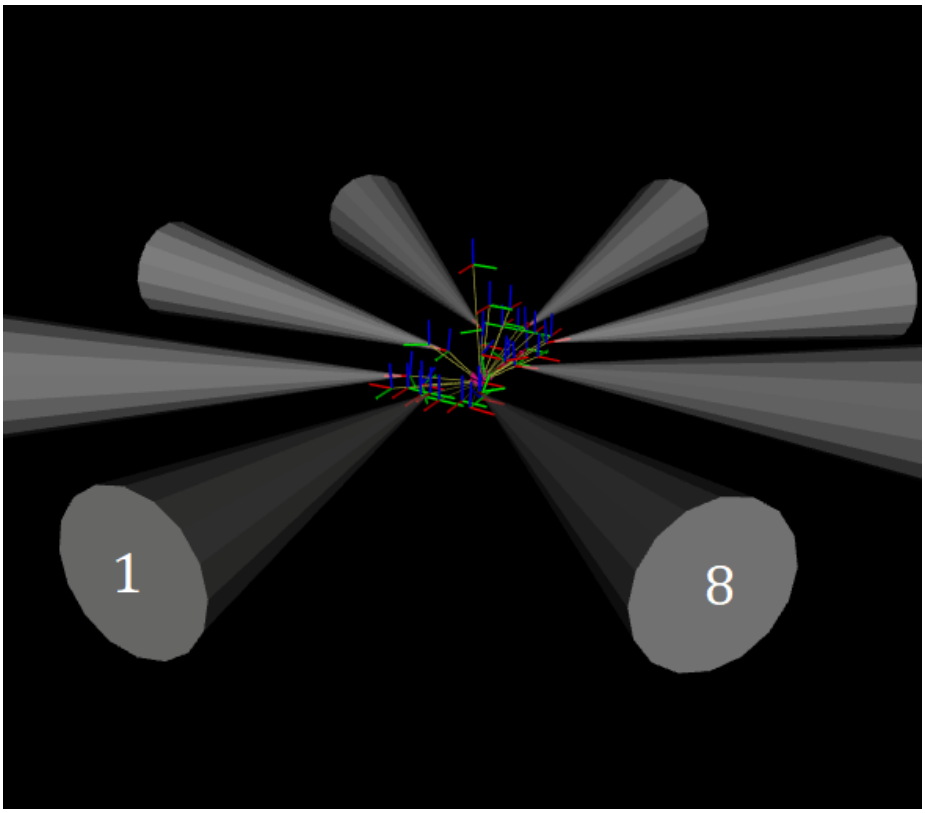

Figure 4.10: Rviz visualization of the applied sensors

Additionally, Figure 4.10 shows the number designated for the first and last sensor implemented. The SONAR number 1 is located in front of the vehicle. The following seven sensors are named in a clockwise way.

Circuit boundaries have been outlined with walls so that proximity sensors could find physical obstacles to guide the vehicle movement. The Prius moves forward as long as the measured distance to the obstacle is superior than a predefined distance value. A defined model of a Sport Utility Vehicle (SUV) is used as obstacle to simulate a vehicle breakdown.

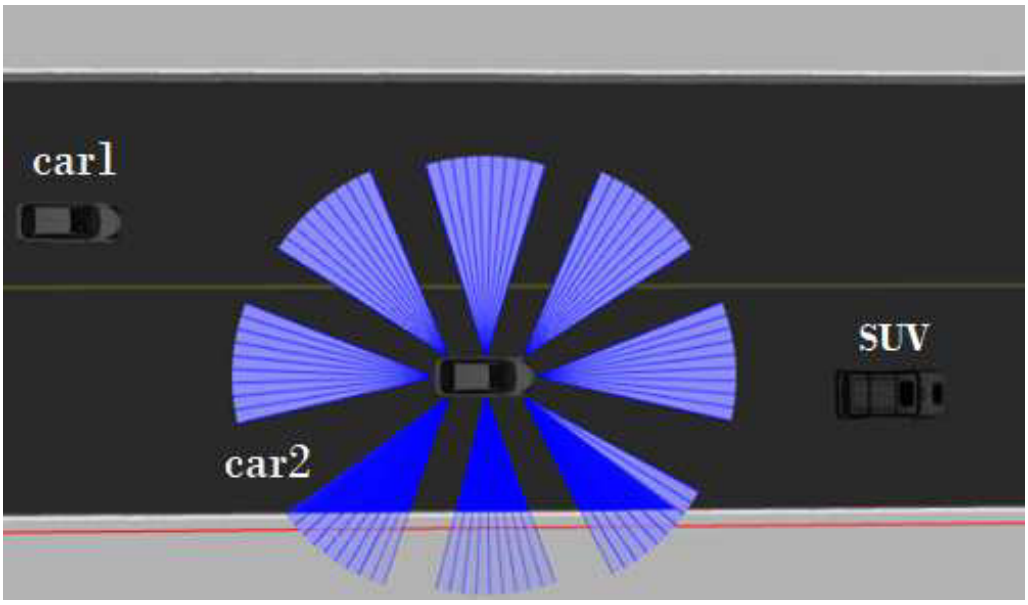

Figure 4.11: Scenario simulation model

Reaching that predefined distance value, the car2 either stops (scenario a) or overtakes the SUV if the left lane is available (scenario b).

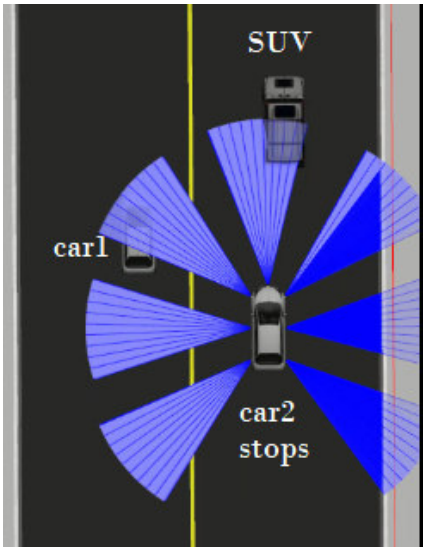

Figure 4.12: Scenario a)

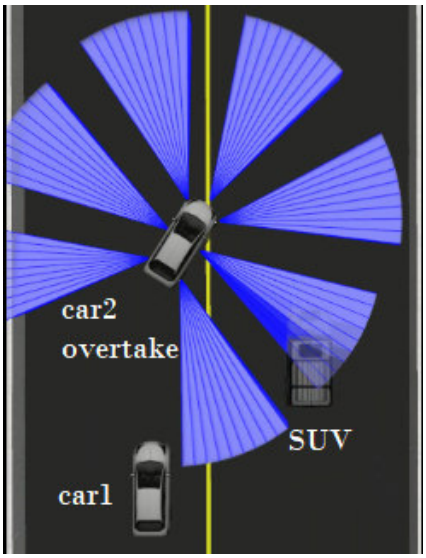

Figure 4.13: Scenario b)

The car2 behavior can be observed in the flowchart represented in Figure 4.14

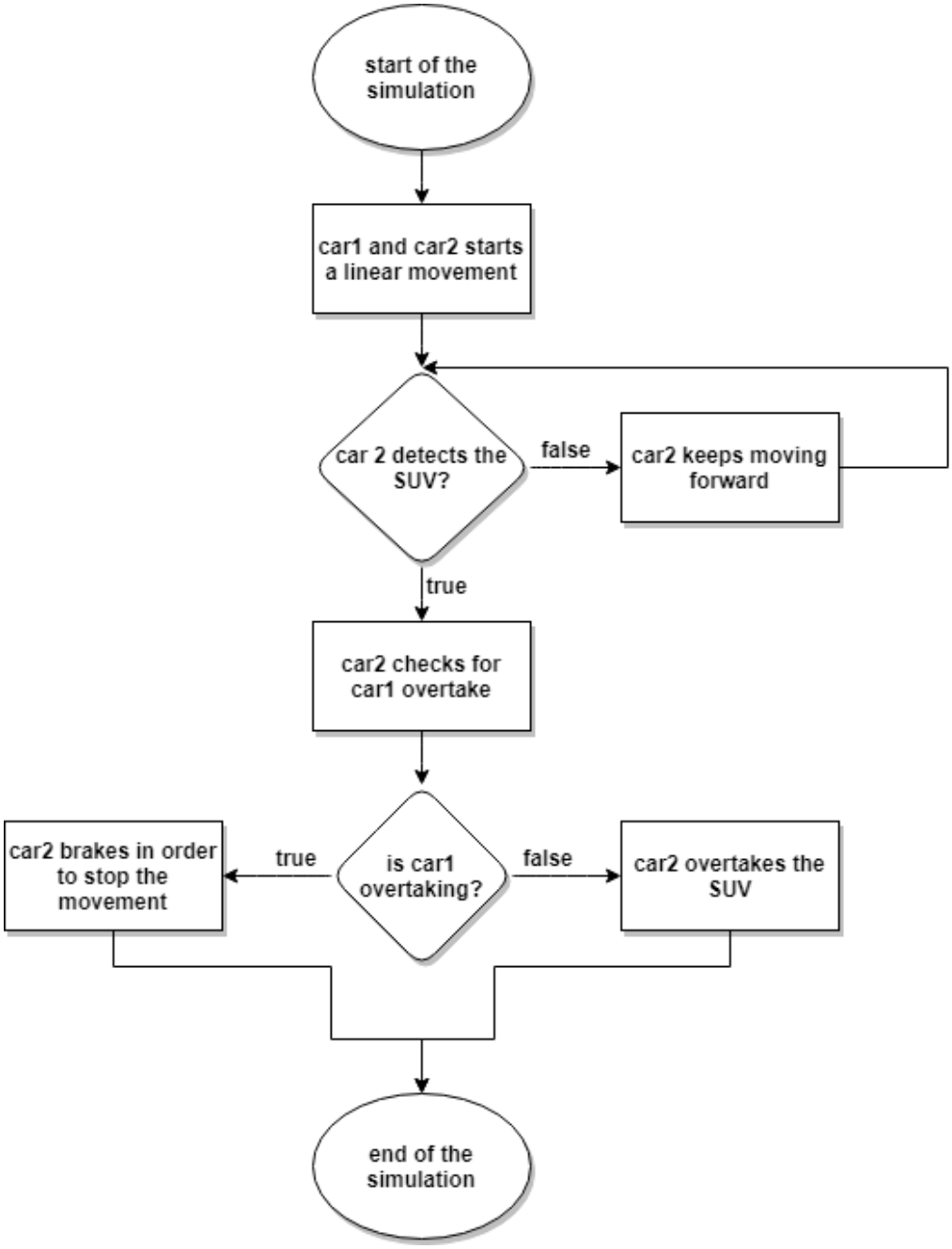

Figure 4.14: Flowchart of the scenario simulation model

4.2.2 ROS2-Gazebo Interface

As mentioned above, ROS2 is still in a preliminary state since the only distribution available (Crystal Clemmys) was released at December 14th, 2018. Therefore, the current design of the integration between Gazebo and ROS2 is not yet complete. The following Table illustrates the package support of both versions of ROS.

ROS integration packages within Gazebo		
ROS version	1	2
gazebo_dev	✓	✓
gazebo_msgs	✓	✓
gazebo_plugins	✓	✓
gazebo_ros	✓	✓
gazebo_ros_control	✓	X
gazebo_ros_pkgs	✓	✓

Table 4.1: ROS packages required to integrate with Gazebo

As depicted in Table 4.1, the control component of the vehicle is not supported by ROS2. Consequently, the previous simulation scenario can not be replicated in this new ROS version.

As a way to overcome this issue, we exploit a package referred as "ros1_bridge [50]. This package is implemented in C++ and provides a network bridge which enables the interchange of messages between ROS1 and ROS2 allowing RO1 tools like Rviz to work with ROS2 applications. The dynamic bridge can automatically create connections while listening to topics from both sides. The message passing can be done through one or both directions.

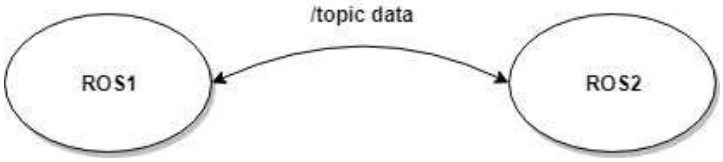

Figure 4.15: Message passing by ros1_bridge

For our use case, the bridge is used to pass the proximity sensor data from ROS1 to ROS2, where the most recent version receives the range from each SONAR and sends a control variable to the oldest ROS version, which uses that data to actuate the vehicle, as shown in Figure 4.16. Wherefore, this system allows ROS2 to use real-time properties to control the car.

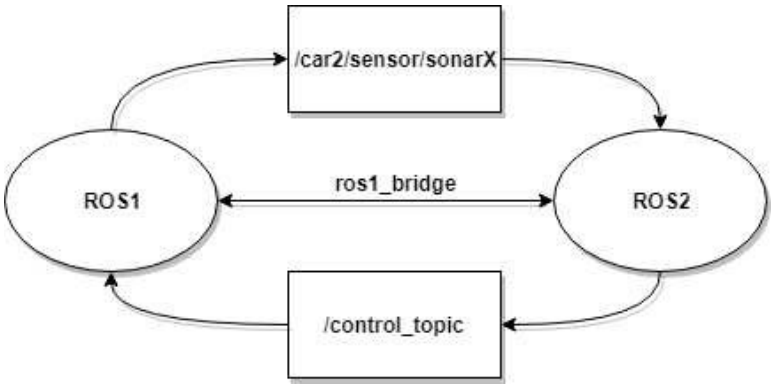

Figure 4.16: Applied system for our use case

4.2.3 Gazebo-OMNeT++ Interface

For our use case, the synchronization between both simulators is a crucial aspect. If the simulation time of both frameworks is asynchronous, the vehicle control and the exchanged messages in OMNeT++ could be compromised, which could lead to a car crash.

OMNeT++ internal clock only works if any simulation event happens, which makes OMNeT++ a discrete-event simulator. Meanwhile, Gazebo has a clock based on time-steps, meaning that the clock works in a fixed frequency according to the defined time-step. Therefore, a methodology to make them synchronized had to be implemented.

As a solution, researchers from CISTER developed a module, named ROSSyncApplication, which provides synchronization methods to associated ROS applications.

The module subscribes to the ROS "Clock" topic, which is published every 1ms, and unlocks the mutex that was previously locked, forcing the OMNeT++ simulation, that was stopped, to advance and schedule new messages to the current ROS/Gazebo timestamps.

Despite the time synchronization between simulators, we used a module, referred as ROSOMNeT, which provides ROS functionalities for module development that might need to use ROS integration in order to fulfill their requirements.

ROSOMNeT provides methods such as rosMain and runROSNode that are quite similar to the ones provided by ROS: the first one is used as the ROS spin loop that usually is associated with ROS applications, and the second one is used to initialize, handle and create a rosNode whenever a module requires it.

For our specific use case, as depicted in Figure 4.18, nine nodes [0-8] were created, in which the node 0 represents the PAN coordinator, and the following ones [1-8] represents each embedded system implemented in the vehicle model, presented in Figure 4.17, respectively.

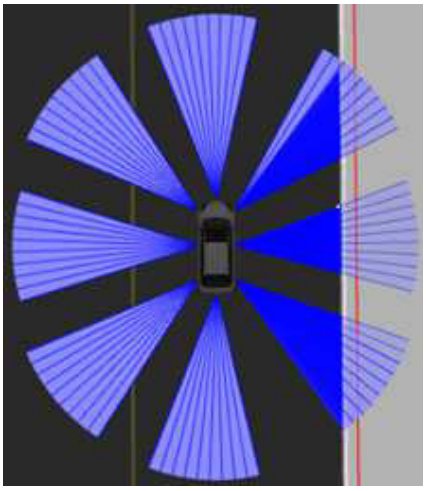

Figure 4.17: ROS nodes

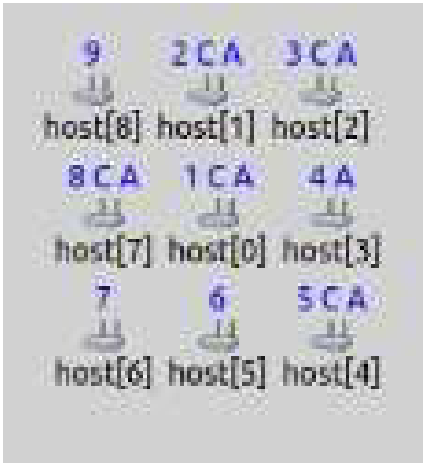

Figure 4.18: OMNeT++ nodes

To provide correct results, the distance between each OMNeT++ node is equal to the distance between each SONAR implemented in the vehicle model. The node number 0 is implemented in the middle of the the eight nodes, therefore, applying this same model into real vehicles forces to place the PAN coordinator into the center of the model.

ROS is the framework base on how both simulators can exchange data. With regard to the particular aspect of how and when data moves around this tool software stack, Figure 4.19 has a understandable overview on it.

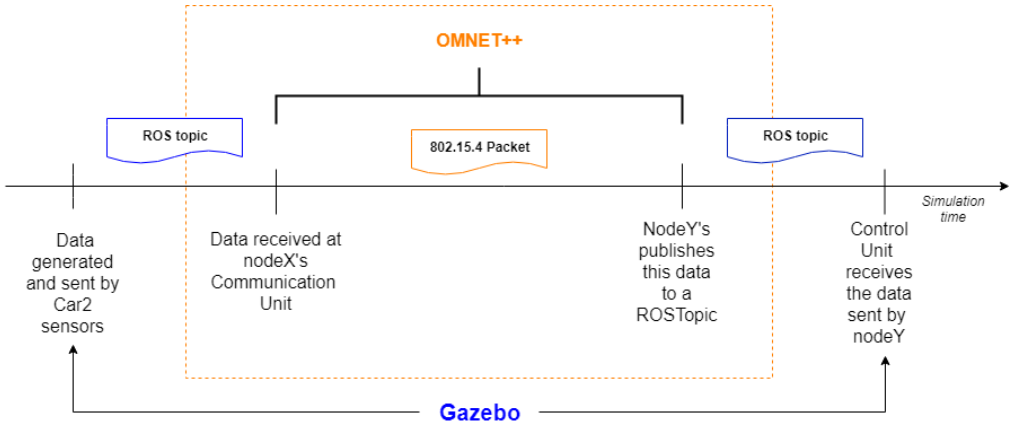

Figure 4.19: Sensor data co-simulation communication

The data from /car2/sensor/sonarX is subscribed by the OMNeT++ nodeX. This node represents the communication unit supporting sonarX. Later, the sensor data, that was transmitted through the DSME communication stack, is published by the PAN Coordinator to a ROS topic. That same topic is subscribed by the control unit of the vehicle.

4.3 Chapter Remarks

The cooperation between the described technologies and tools used in the simulation scenario allows us to keep on track the behavior of each implemented sensor and the movement of the vehicle. Adding to this, with the implementation of the network simulation framework, the vehicle model behaves in a different way, since there is delay associated with the message and data providing.

Therefore, this system allows us to extract data concerning the performance of the network, along with the behaviour of ROS2 as a middleware for ADAS applications. This data analysis is done in the next chapter.

Chapter 5

Performance Analysis of ADAS

This chapter presents the performance analysis of the IEEE 802.15.4 DSME for ADAS. Moreover, the configurations of the MAC behavior properties to test the effectiveness of the network in the vehicle behavior.

5.1 Co-Simulation Performance Analysis

To test the network impact on our use case, we test the simulation performance when supported by the sensitive network DSME pre-set (BO=6, SO=4, MO=6). Figure 5.1 shows the percentage of crashes, as the car full brakes when the range value is smaller than the maximum SONAR range (9 meters), on a five trial data set, as the speed is increased.

Through graph analysis, we conclude that the maximum speed for this scenario is 24.5 km/h. This is the maximum speed the car2 can achieve, without compromising the safety guarantees of the vehicle. Increasing the maximum speed would expand the risk of collision, since the car stops closer to the SUV. At speed of 30 km/h, a crash would be inevitable.

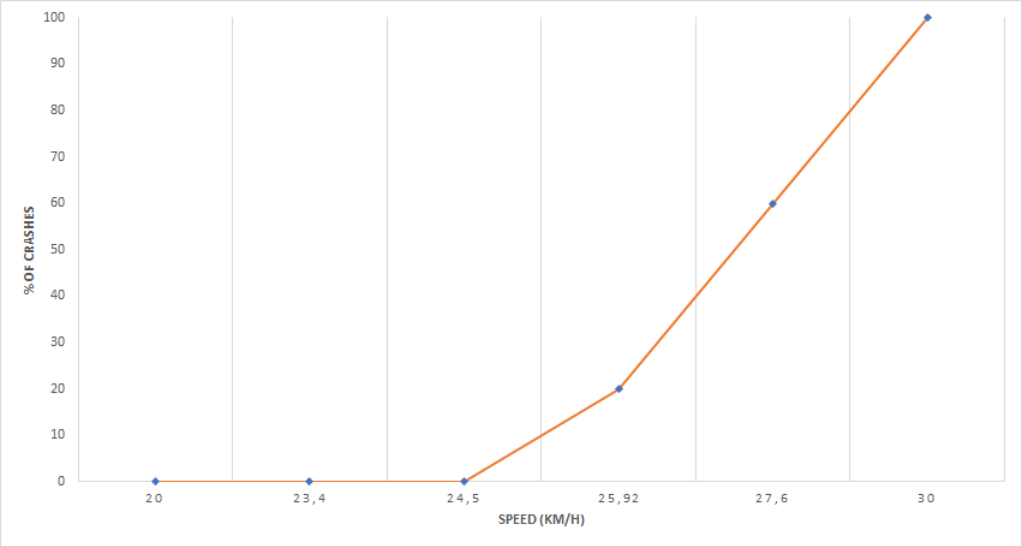

Figure 5.1: Percentage of crashes as the speed is increased (MO=6)

At the maximum speed, the car2 stops, as depicted in Figure 5.2, 71 cm behind the SUV. This distance enables to immobilize the vehicle in such a way as to prevent an accident. However, it does not allow to overtake the obstacle.

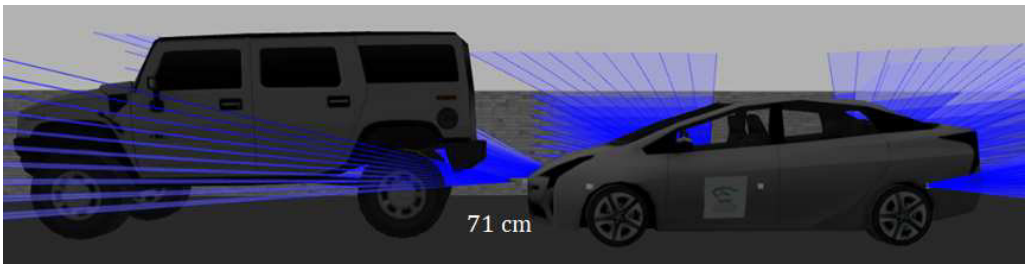

Figure 5.2: Minimum distance brake

The introduction of a network in similar ADAS creates a communication delay between the system control unit and the communication unit. The main goal of alike safety critical systems is to get as close as possible to perfect systems: systems without any kind of failure.

Although these systems do not yet exist, and most likely will never exist, the DSME MAC behavior aims to reduce that same delay in order to prevent failures in real time based applications.

Particularly in scenarios as ADAS, where timeliness is a crucial component, the delay should be as close as possible to 0. For instance, a vehicle that moves at a speed of 50 km/h with a delay of 1s while braking, will stop 14 meters ahead of where it should.

To calculate the delay of all the sensors that are involved in the simulation, we test the efficiency of the scenario a). The speed measurement of car2 is made through a topic referred as /carX/carINFO, which provides data regarding the speed, gas and brake pedal percentage, steering angle and GPS coordinates.

5.2 Network Performance Analysis

Our system was designed to keep on track the time interval between the sending and receiving of each packet between both application layers. Nevertheless, this grants us the possibility of calculating the associated delay for each packet transmission.

However, since the SONAR's are transmitting data at 5Hz frequency, the simultaneous transmission of the eight sensors provides too many packets for the network to handle. As a way to overcome this issue, we designed a system where, even though all the range values from each sensor are sent to the PAN coordinator, the OMNeT++ communication unit only pass the range value to the ROS control unit when the current value differs from a defined interval regarding the previous one. This relevance system allows us to still keep on track the behavior of each sensor and, most importantly, reduce packet traffic.

In this scenario, only four sensors have to pick data from the stationary SUV (SONAR1) and from the car1 overtake (SONAR's 6,7,8). Figure 5.3 shows the packet sending and receiving on a time based scale. The network configurations used for this simulation were (BO=6, SO=4, MO=6), which is the network pre-set for sensitive networks.

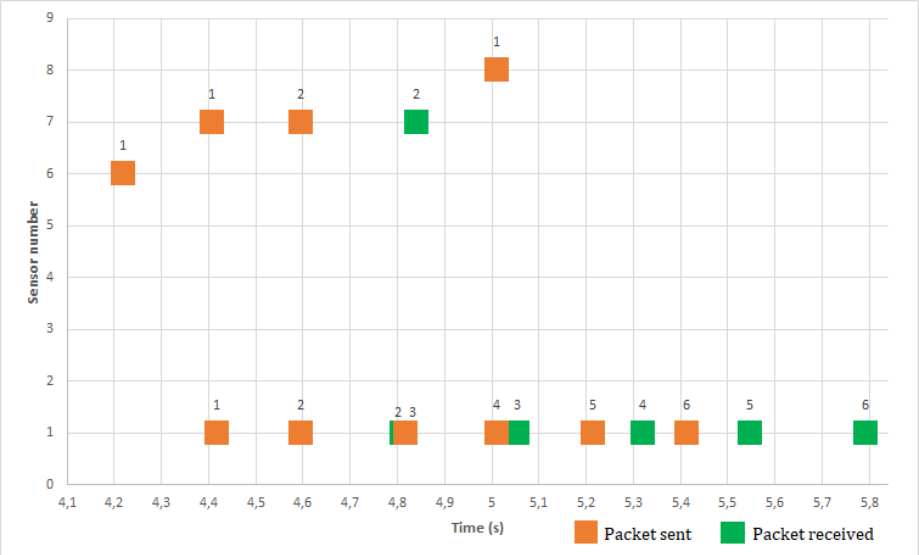

Figure 5.3: Nodes packet passing

From graph interpretation, it is possible to observe that the maximum transmission delay is 0,4s (packet 6 from node 6) and the minimum one is 0,22s (packet 2 from node 6).

As said before, the car2 maximum speed for this network pre-set is 24.5 km/h. However, only the delay analysis of each sensor does not give us any insight into its relationship to the vehicle's behavior against obstacles.

Therefore, four trials were conducted to obtain the delay of each sensor at the same maximum speed. Figure 5.4 shows the delay of each sensor according to the maximum acceptable delay. The maximum acceptable delay is determined through the range of the SONAR and the vehicle speed. According to our experiments, the maximum acceptable delay is 1s.

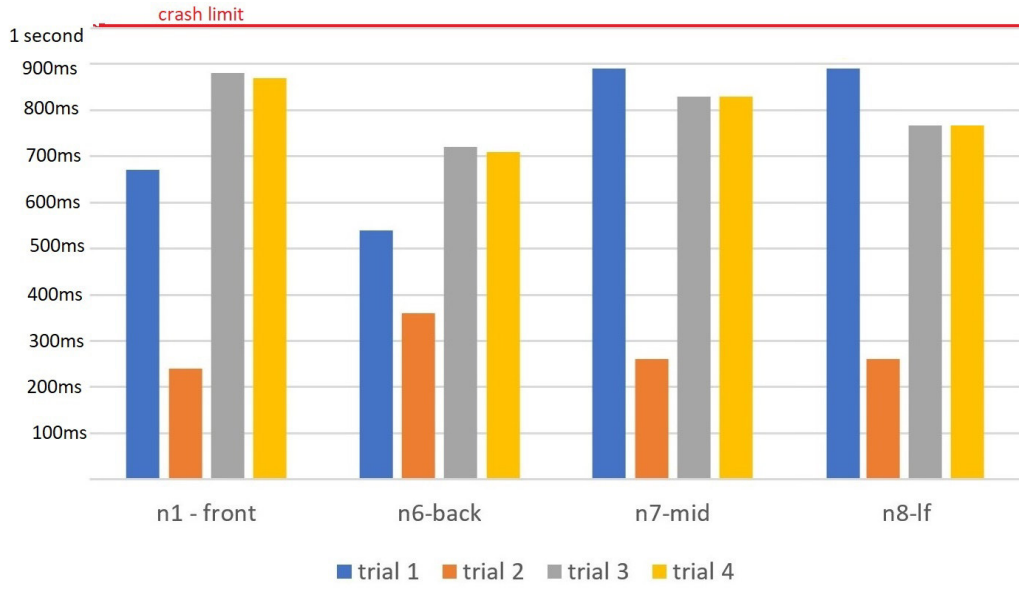

Figure 5.4: Packet delay of each sensor

According to Chapter 2, the overall network QoS can be increased by changing the value of MO. Nevertheless, the CAP Reduction can also lower the communication delay. However, in the particular case of ADAS, since the data is being constantly delivered at a high frequency, activating the CAP Reduction mechanism would increase the overall bandwith but will not be the best choice for a fast and reliability oriented application like intra-car ADAS.

MO can also have an similar impact to the CAP Reduction. Larger MOs result in many superframes inside one multi superframe duration. Consequently, nodes wait for several superframe durations to send the respective data frames, increasing delay.

Three trials were conducted to calculate the SONAR1 delay incurred in the data transmission for various MO settings. Only this sensor was tested since it is the only one that detects the SUV and starts the process. Figure 5.5 shows the delay results when reducing the MO value from the previous delay sensitive settings (BO=6, SO=4, MO=6).

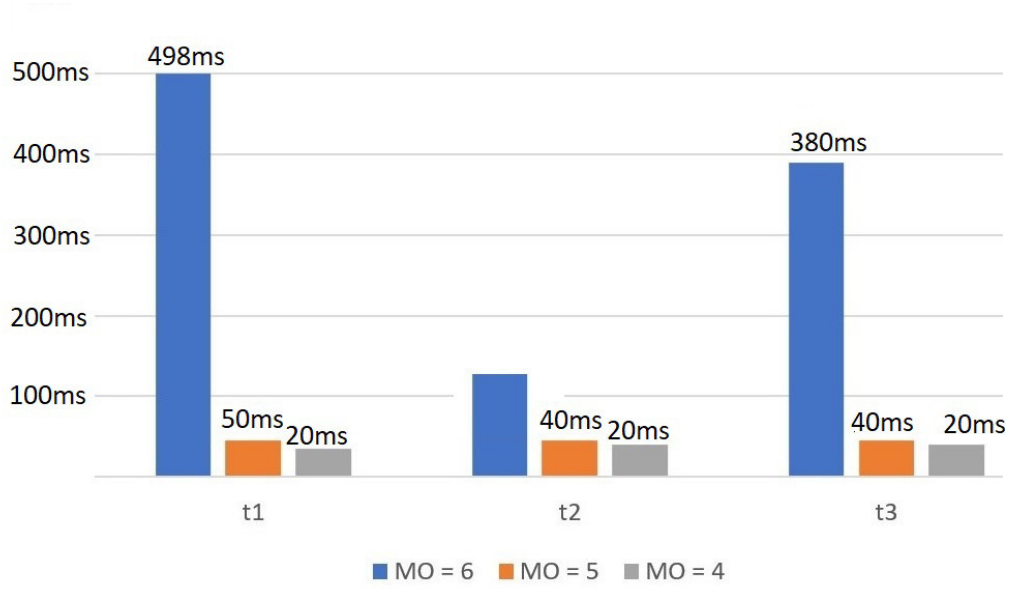

Figure 5.5: Nodes packet passing

Through graph interpretation, the delay is reduced by decreasing the MO value. In the first trial (MO=6), four superframes were encompassed within a single multi superframe duration. Each node is obligated to wait several superframe duration's to transmit the data that is being constantly published at a high frequency. Naturally, the PAN coordinator receives the data with an associated delay. Even so, in the third trial (MO=4) the delay is extraordinary lesser, obtaining a minimal transmittal delay of 20 ms.

5.2.1 Application Impact

To test the network impact on our use case, we test the simulation performance when supported by the sensitive network DSME pre-set (BO=6, SO=4, MO=6), and by the best results obtained when reducing the MO value (BO=6, SO=4, MO=4).

For a smaller MO value, Figure 5.6 shows the percentage of crashes in the exact same conditions previously tested (blue) along with the previous obtained results (orange).

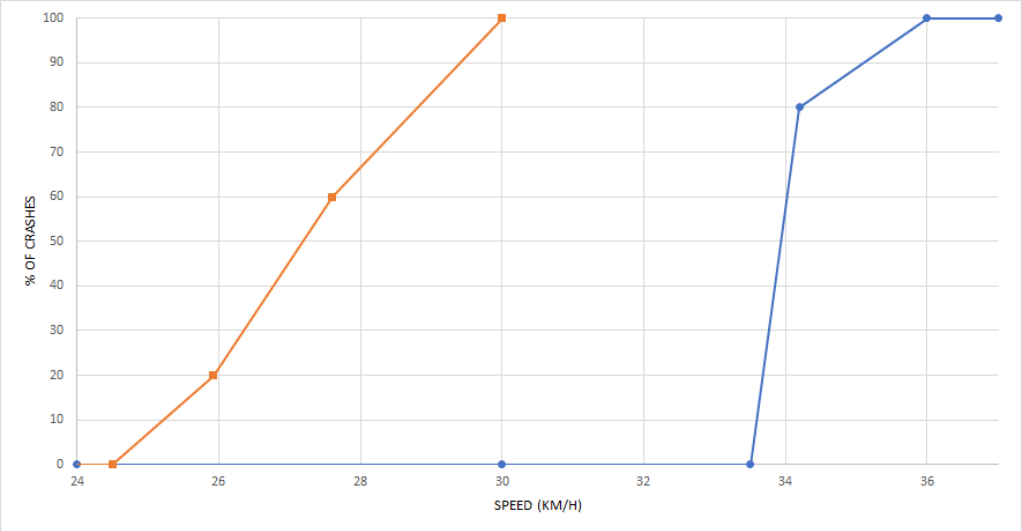

Figure 5.6: Percentage of crashes as the speed is increased for different MO values

Naturally, it is possible to deduce that he shorter the delay, the higher the maximum speed and more linear the crash and no crash ratio becomes. This is due to higher speeds, which are translated to bigger braking distances. Moreover, within this DSME configuration settings, the maximum speed is 33.5 km/h, which is 9 km/h higher than the previous test. This is a significant difference in terms of application behavior.

Additionally, as depicted in Figure 5.7, the minimum brake distance between the car and the SUV was 82cm, which is bigger than the previous distance since the Prius moves at a higher speed.

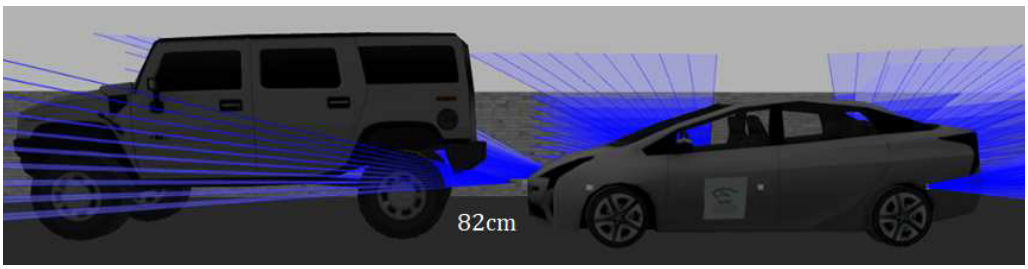

Figure 5.7: Minimum distance brake (MO=4)

Chapter 6

Conclusions

This chapter evaluates the proposed objectives of this thesis and discusses the obtained results, along with the research contributions provided. This chapter presents an overview for the future work that must be developed according to the established objectives.

6.1 Obtained Results

Considering the initial objectives proposed for this thesis, it is concluded that, although it is a work in progress, the work developed in this thesis allowed significant advances for the wireless retrofitting ADAS in older vehicles.

With the results obtained through the inter cooperation of the various simulators used throughout this thesis, we were able to analyze the DSME MAC behavior performance and its reliability when supporting similar safety critical systems. By analyzing the associated delay when changing the DSME parameters and the respective application behavior, we conclude that the standard provides a acceptable delay for safety critical systems.

This same conclusion is based on the type of application that DSME supports. The range of the sensors used in this scenario were designed in a way to test the network communication capabilities. Immobilizing a vehicle that moves with a speed of 33.5 km/h by only depending of a 9m sensor range, we can conclude that this same standard can be applied in real vehicles, since there are sensors with more range and more suitable properties for similar systems.

6.2 Future Work

Primarily, there are some issues, not directly related to the objectives of this thesis, that needs to be addressed. First, as can be deduced by analyzing the obtained results, the delay values, in the same exact conditions, are unpredictable and inconstant. For this reason, simulate the scenario b) with a network support is not possible. This is due to the scheduler algorithm implemented in the simulation framework, whose behavior does not meet with the intended. Consequently, slot allocation is not done in a predictable way, which affects the delay in a direct way, since some packets must wait for the next superframe to be transmitted. This is a crucial aspect when it comes to safety critical systems, in which should be referred as future research work.

Secondly, despite the fact that the scripts were already developed by us and the bridge allows ROS2 to subscribe each sensor data, it is not possible, yet, to test them in our scenario. This is due to a conflict between the OS supported by the OMNeT++ and ROS, which are Ubuntu 16.04 and Ubuntu 18.04, respectively. Therefore, as future work, we intend to create a communication system between these two frameworks, which would allow us to take conclusive results regarding the ROS2 real-time properties, when supported by a WSN, for safety critical systems such as ADAS.

References

[1] "reduced-function device - an overview | sciencedirect topics." https://www.sciencedirect.com/topics/computer-science/reduced-function-device. (Accessed on 08/27/2019). [cited on p. v, 6, 9]

[2] W. D. S. Morgan Quigley, *Programming Robots with ROS*. O'Reilly, 2015. [cited on p. v, 8, 24]

[3] "Moos : Main - introduction browse." http://www.robots.ox.ac.uk/~mobile/MOOS/wiki/pmwiki.php/Main/Introduction. (Accessed on 07/4/2019). [cited on p. v, 12, 19, 20, 21, 24]

[4] S. Tadakamadla and B. Oelmann, "Indoor local positioning system for zigbee, based on rssi," 01 2006. [cited on p. v, 14]

[5] Y. Al-Nidawi, H. Yahya, and A. H. Kemp, "Impact of mobility on the iot mac infrastructure: Ieee 802.15.4e tsch and lldn platform," in *2015 IEEE 2nd World Forum on Internet of Things (WF-IoT)*, pp. 478–483, Dec 2015. [cited on p. v, 12, 15, 16, 18]

[6] M. Quigley, K. Conley, B. P Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y Ng, "Ros: an open-source robot operating system," vol. 3, 01 2009. [cited on p. v, 25, 26, 27, 28, 29, 31, 34]

[7] "Rti connext dds : Setup and helloworld example (windows/eclipse/java) | service engineering (icclab & splab)." https://blog.zhaw.ch/icclab/rti-connext-dds-setup-and-helloworld-example-windowseclipsejava/. (Accessed on 08/27/2019). [cited on p. v, 30]

[8] "Robótica open source: Começando com gazebo e ros 2." https://www.infoq.com/br/articles/ros-2-gazebo-tutorial/. (Accessed on 08/27/2019). [cited on p. v, 32]

[9] "Piloto automático | tesla." https://www.tesla.com/pt_PT/autopilot. (Accessed on 08/28/2019). [cited on p. v, 37]

[10] "osrf/car_demo." https://github.com/osrf/car_demo. (Accessed on 08/27/2019). [cited on p. vi, 38]

[11] "urdf/xml/link - ros wiki." http://wiki.ros.org/urdf/XML/link. (Accessed on 08/10/2019). [cited on p. vi, 39]

[12] "urdf/xml/joint - ros wiki." http://wiki.ros.org/urdf/XML/joint. (Accessed on 08/10/2019). [cited on p. vi, 40]

[13] V. Hax, N. Duarte Filho, S. Botelho, and O. Mendizabal, "Ros as a middleware to internet of things," *Journal of Applied Computing Research*, vol. 2, pp. 91–97, 07 2013. [cited on p. 1]

[14] "Who | world health organization." https://www.who.int/. (Accessed on 08/27/2019). [cited on p. 1]

[15] J. Harrison Kurunathan, "Improving qos for ieee 802.15.4e dsme networks," pp. 91–97, 11 2016. [cited on p. 2, 5, 6, 8, 9]

[16] T. Radivilova, Y. Ibrahim Daradkeh, and L. Kirichenko, "Development of qos methods in the information networks with fractal traffic," *International Journal of Electronics and Telecommunications*, vol. 64, pp. 27–32, 01 2018. [cited on p. 2]

[17] "pt - ros wiki." http://wiki.ros.org/pt. (Accessed on 08/27/2019). [cited on p. 2]

[18] Y. Maruyama, S. Kato, and T. Azumi, "Exploring the performance of ros2," *2016 International Conference on Embedded Software (EMSOFT)*, pp. 1–10, 2016. [cited on p. 2, 28, 30]

[19] G. A. Domenico De Guglielmo, Simone Brienza, "Ieee 802.15.4e: a survey," pp. 1–10, 2016. [cited on p. 5, 19]

[20] "Energy conservation in wireless sensor networks: A survey," [cited on p. 5]

[21] H. Kurunathan, R. Severino, A. Koubaa, and E. Tovar, "Ieee 802.15.4e in a nutshell: Survey and performance evaluation," *IEEE Communications Surveys Tutorials*, vol. 20, pp. 1989–2010, thirdquarter 2018. [cited on p. 5, 9, 10, 11, 15, 18, 20]

[22] "Part 15.4: Wireless medium access control (mac) and physical layer (phy)specifications for low-rate wireless personal area networks (wpans)," [cited on p. 6, 7, 8]

[23] R. A. R. Silva Severino, "On the use of ieee 802.15.4/zigbee for time-sensitive wireless sensor network applications," pp. 0–43, 11 2008. [cited on p. 6, 8, 9, 10, 19, 20]

[24] "Personal area networks - optical zeitgeist laboratory." `http://www.zeitgeistlab.ca/doc/personal-area-networks.html`. (Accessed on 08/27/2019). [cited on p. 7, 8]

[25] A. Sheraz, W. Khan, J. Bangash, S. irfan ullah, A. Salam, A. Khan, and S. Ahmed, "Impact of beacon order and superframe order on ieee 802.15.4 for nodes association in wban," *EAI Endorsed Transactions on Energy Web*, vol. 5, 01 2018. [cited on p. 9]

[26] Rudiyanto and R. F. Sari, "Analysis of the effect of beacon order and superframe order value to the performance of multihop wireless networks on ieee 802.15.4 protocol," in *2012 International Conference on Advanced Computer Science and Information Systems (ICACSIS)*, pp. 89–94, Dec 2012. [cited on p. 9]

[27] M. S. Xia Cheng, Junyang Shi, "Cracking the channel hopping sequences in ieee 802.15.4e-based industrial tsch networks," 04 2019. [cited on p. 15]

[28] "Ros/introduction - ros wiki." `http://wiki.ros.org/ROS/Introduction`. (Accessed on 08/27/2019). [cited on p. 24]

[29] "Carmen." `http://carmen.sourceforge.net/`. (Accessed on 07/4/2019). [cited on p. 24]

[30] "Player project." `http://playerstage.sourceforge.net/`. (Accessed on 07/4/2019). [cited on p. 24]

[31] "Why choose ros? [closed] - ros answers: Open source q&a forum." `https://answers.ros.org/question/33811/why-choose-ros/`. (Accessed on 06/29/2019). [cited on p. 24]

[32] M. Bawa, B. F. Cooper, A. Crespo, N. Daswani, P. Ganesan, H. Garcia-Molina, S. Kamvar, S. Marti, M. Schlosser, Q. Sun, P. Vinograd, and B. Yang, "Peer-to-peer research at stanford," pp. 1–10, 2016. [cited on p. 25]

[33] "Ros/technical overview - ros wiki." `http://wiki.ros.org/ROS/Technical%20Overview`. (Accessed on 08/27/2019). [cited on p. 27]

[34] "Why ros 2?." `http://design.ros2.org/articles/why_ros2.html`. (Accessed on 07/15/2019). [cited on p. 29]

[35] S. Profanter, A. Tekat, K. Dorofeev, M. Rickert, and A. Knoll, "Opc ua versus ros, dds, and mqtt: Performance evaluation of industry 4.0 protocols," 02 2019. [cited on p. 29, 30]

[36] O. M. G. (OMG), "Data distribution service (dds)," 04 2015. [cited on p. 30]

[37] "Gazebo." `http://gazebosim.org/`. (Accessed on 08/27/2019). [cited on p. 31]

[38] K. Takaya, T. Asai, V. Kroumov, and F. Smarandache, "Simulation environment for mobile robots testing using ros and gazebo," in *2016 20th International Conference on System Theory, Control and Computing (ICSTCC)*, pp. 96–101, Oct 2016. [cited on p. 31]

[39] B. F. B. Barros Vieira, "A simulation approach for increased safety in advanced c-its scenarios," pp. 0–43, 11 2008. [cited on p. 31, 35]

[40] "Visualize with rviz | learn ubiquity robots and ros." `https://learn.ubiquityrobotics.com/rviz`. (Accessed on 08/27/2019). [cited on p. 32]

[41] E. Anggadjaja and I. Mcloughlin, "Point-to-point omnet++ based simulation of reliable transmission using realistic segmentation and reassembly with error control," in *2010 Second International Conference on Advances in Computing, Control, and Telecommunication Technologies*, pp. 125–128, Dec 2010. [cited on p. 33]

[42] Z. K. P. S. Radek Silhavy, Roman Senkerik, *Software Engineering Perspectives and Application in Intelligent systems*. PACKT, 2015. [cited on p. 33]

[43] A. Varga and R. Hornig, "An overview of the omnet++ simulation environment," in *Proceedings of the 1st International Conference on Simulation Tools and Techniques for Communications, Networks and Systems & Workshops*, Simutools '08, (ICST, Brussels, Belgium, Belgium), pp. 60:1–60:10, ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2008. [cited on p. 33]

[44] "opendsme." `http://opendsme.org/`. (Accessed on 08/27/2019). [cited on p. 35]

[45] M. Köstler, F. Kauer, T. Lübkert, and V. Turau, "Towards an open source implementation of the ieee 802 . 15 . 4 dsme link layer," 2016. [cited on p. 35]

[46] "Cometos/cometos: A component-based, extensible, tiny operating system for wireless networks." `https://github.com/CometOS/CometOS`. (Accessed on 08/27/2019). [cited on p. 35]

[47] A. U. Yousuf, W. Lehman, M. A. Mustafa, and M. M. A. Hayder, "Introducing kinematics with robot operating system (ros)," 2015. [cited on p. 39]

[48] A. Boyat and B. Joshi, "A review paper: Noise models in digital image processing," *Signal Image Processing : An International Journal*, vol. 6, 05 2015. [cited on p. 42]

[49] A. ADEMOVIC, "An introduction to robot operating system | toptal." `https://www.toptal.com/robotics/introduction-to-robot-operating-system`. (Accessed on 07/22/2019). [cited on p. 45]

[50] "ros2/ros1_bridge: Ros 2 package that provides bidirectional communication between ros 1 and ros 2." `https://github.com/ros2/ros1_bridge`. (Accessed on 08/27/2019). [cited on p. 48]

Anexo A

Project Roadmap