

Evaluating 802.11p Communications Performance for a Safety-Critical Vehicular Application

João Filipe Mateus Pereira

LEEC

Licenciatura em Engenharia Eletrotécnica e de Computadores



DEPARTAMENTO DE ENGENHARIA ELETROTÉCNICA
Instituto Superior de Engenharia do Porto

Setembro, 2021

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: João Filipe Mateus Pereira, N.º 1180526, 1180526@isep.ipp.pt

Orientação Científica: David Miguel Ramalho Pereira, drp@isep.ipp.pt

Empresa: CISTER-Research Center of Real-Time & Embedded Computing Systems

Orientador: Pedro Miguel Salgueiro Santos, pss@isep.ipp.pt



DEPARTAMENTO DE ENGENHARIA ELETROTÉCNICA
Instituto Superior de Engenharia do Porto
Rua Dr. António Bernardino de Almeida, 431, 4200-072 Porto

Setembro, 2021

Para a minha família e amigos . . .

Agradecimentos

Nesta secção, gostaria de agradecer ao meu orientador Eng. Pedro Santos, ao Eng. Ênio Filho e ao Eng. David Pereira pela orientação relativa ao trabalho, sem eles nada seria possível. Considerando isso, agradeço à minha família e amigos que indiretamente me empurram para frente, para assim, eu conseguir fazer o melhor para a minha vida. Este projeto ajudou-me a crescer e fico feliz por o poder dizer. Esta tese, termina perfeitamente este ciclo e espero que inicie um novo, com muitos mais desafios pela frente. Concluindo esta secção, gostaria de agradecer à minha namorada, Mariana Carneiro, por estar ao meu lado em todos os momentos e me dar forças para continuar e nunca desistir até que tudo esteja no caminho que deve estar. A todos os meus colegas do CISTER, agradeço a liberdade e o bem-estar que me proporcionaram no tempo que aqui estive. A todos, obrigado!

Resumo

A humanidade, está a caminhar, progressivamente, para um futuro mais intuitivo e tecnológico. A área dos Sistemas Inteligentes e Cooperativos de Transporte tem-se revelado como uma das áreas em grande evolução, através de tecnologias de condução autónoma e comunicação intra-veicular. O projeto CMU|PT FLOYD tem como objetivo explorar essa mesma área e desenvolver novos sistemas de possível implementação futura.

Já neste relatório, é efetuada a avaliação de um cenário de *Emergency Braking*, com a implementação de *Decentralized Environmental Notification Messages* (DENM) para criação de um Sistema de Travagem de Emergência (STE). Neste mesmo caso, é também desenvolvido um cenário de *platooning* focado na comunicação *Vehicle to Vehicle* (V2V) e *Vehicle to Infrastructure* (V2I). Esta comunicação é realizada utilizando a norma ETSI ITS-G5, que é a norma-padrão a nível europeu relativamente a comunicações veiculares.

Neste projeto, foram desenvolvidos componentes de *software* que permitem uma integração em tempo real entre dois simuladores, um deles relacionado com a rede de comunicação e outro com a parte robótica e simulada de todo o cenário. Esta integração realizou-se a partir da adaptação de algumas ferramentas já existentes de forma a poder ser implementado o cenário de um STE.

Por fim, uma validação ao sistema foi realizada, com o uso de diversas *velocidades-target* e frequências para perceber melhor as limitações do cenário e do STE. A grande maioria dos resultados apresentados referem-se ao pelotão e ao comportamento do mesmo perante as diferentes condições de travagem.

Palavras-Chave: ROS, V2V,V2I, Pelotão de Veículos, Simulação, Sistemas Inteligentes e Cooperativos de Transporte.

Abstract

Humanity is progressively moving towards a more intuitive and technological future. The area of Cooperative Intelligent Systems and Transport has proved to be one of the areas in great evolution, through autonomous driving technologies and intra-vehicle communication. The CMU|PT FLOYD project aims to explore this same area and develop new systems for possible future implementation.

In this report, an *Emergency Braking* scenario is evaluated, with the implementation of DENM to create an STE. In this same case, a *platooning* scenario focused on V2V and V2I communication is also developed. This communication is carried out using the ETSI ITS-G5 standard, which is the European standard for vehicular communications.

In this project, *software* components were developed that allow a real-time integration between two simulators, one of them related to the communication network and the other with the robotic and simulated part of the entire scenario. This integration was carried out by adapting some existing tools so that the scenario of an STE could be implemented.

Finally, a validation of the system was performed, using different target speeds and frequencies to better understand the limitations of the scenario and the STE. The vast majority of the results presented refer to the peloton and its behavior under different braking co

Keywords: *Cooperative Intelligent Transport System (C-ITS), V2V, V2I, Robotic Operation System (ROS), Cooperative Platooning, Simulation.*

Índice

Lista de Figuras	vii
Lista de Tabelas	ix
Lista de Acrónimos	xi
1 Introdução	1
1.1 Motivação	1
1.2 Contextualização	2
1.3 Objetivos e Descrição do Projeto	2
1.4 Calendarização	3
1.5 Organização do Relatório	3
2 Comunicações Veiculares - Tecnologias	5
2.1 Arquitetura C-ITS	5
2.2 Visão Geral das Tecnologias WAVE e ITS-G5	7
2.3 Protocolo IEEE 802.11p	8
2.3.1 Camada MAC	9
2.3.2 Camada Física	10
2.3.3 Vantagens	10
2.4 ETSI ITS-G5	10
2.4.1 Applications	12
CAM Messages	13
DENM Messages	14
2.4.2 Security	15
2.4.3 Networking e Transport	16
2.4.4 Camada de Acesso	17
2.4.5 Management	17
2.4.6 Facilities	17
3 Tecnologias de Simulação	19
3.1 OMNet++	19
3.2 Artery	21
3.3 Gazebo	23

3.4	Robot Operation System (ROS)	24
3.5	COPADRIVE	26
3.6	<i>Setup</i> de Simulação	30
4	Sistema de Travagem de Emergência com Pelotões Cooperativos	33
4.1	Pelotões Cooperativos	33
4.2	Sistema de Travagem de Emergência (STE)	35
4.3	Cenário STE - Descrição	36
5	Implementação do STE	41
5.1	Sistema de Travagem de Emergência - Estrutura geral	41
5.2	Adaptação do ficheiro DenService	43
5.3	STE - Especificação e Funcionalidades	45
5.4	Ligação ROS/Gazebo - Paragem dos veículos	48
5.5	Conclusões e pontos a salientar	49
6	Validação e Resultados Finais	51
6.1	Descrição da Experiência	51
6.2	Resultados e Validação do Cenário-Principal	54
6.3	Resultados para diferentes velocidades	55
6.4	Resultados para diferentes frequências de envio de mensagens	58
6.5	Conclusões e pontos a salientar	58
7	Conclusões	61
7.1	Trabalho Futuro	62
	Referências	63

Lista de Figuras

1.1	Planeamento de Projeto	3
2.1	Cenário - VANET [1]	6
2.2	C-ITS - Tipos de ligação numa rede VANET [2]	7
2.3	WAVE e ITS-G5 - Comparação [3]	8
2.4	Exemplo - Arquitetura WAVE (EUA)	9
2.5	Rede de Sistemas ITS	11
2.6	Distribuição - Estrutura ETSI [4]	12
2.7	CAM - CaService [5]	13
2.8	DENM - DenService [6]	15
2.9	Security - Funcionalidades [2]	15
2.10	Networking e Transporte - Principais pontos [2]	17
2.11	Management - parâmetros [2]	17
2.12	Facilities - Funcionalidades [2]	18
3.1	Estrutura por Módulos - OMNeT++ [2]	20
3.2	OMNeT++ - Interface Gráfico	21
3.3	Estrutura - Artery [7]	22
3.4	Interface Gazebo	23
3.5	Veículo Utilizado	24
3.6	Cenário - Interseção	24
3.7	ROS - Publicação/Subscrição [8]	25
3.8	ROS - Exemplo [9]	26
3.9	COPADRIVe - Estrutura [10]	26
3.10	Vehicle Data Provider - RXNetCallback()	27
3.11	CaService - createCooperativeAwarenessMessage()	28
3.12	Robot Middleware - ReceiveSignal()	28
3.13	ROSSyncApplication - handleMessage()	29
3.14	COPADRIVe - Estrutura [10]	29
3.15	platooning.py - Distance-Control()	30
3.16	Diagrama - Setup Simulação	31
4.1	Cooperative Platooning - Exemplo [11]	34
4.2	Componentes do STE e Comunicação entre dispositivos	35

4.3	Arquitetura Geral STE	36
4.4	Prioridades - Interseção [12]	37
4.5	Cenário Emergency Braking	38
5.1	Diagrama de Sequência da implementação do STE	42
5.2	Diagrama Temporal - Envio de Mensagens CAM e DENM	43
5.3	DenService - Chamada do Timer	44
5.4	DenService - VerifyStation()	44
5.5	DenService - SendDenm()	45
5.6	DenService - Trigger()	45
5.7	UseCaseRSU - CreateMessageSkeleton()	46
5.8	EmergencyBraking- CreateMessage()	47
5.9	EmergencyBraking - CreateRequest()	47
5.10	LaneDetection - callback()	48
5.11	Simulation-Connector.py - EmergencyBraking()	49
6.1	Espectro - DSRC [13]	51
6.2	EmergencyBraking - Check()	52
6.3	Envio das mensagens - OMNeT++	53
6.4	Emergency Braking - Cenário	54
6.5	Velocidades do Car1 e Car2 a 50 km/h	54
6.6	Velocidades do Car3 e Car4 a 50 km/h	55
6.7	Análise da distância entre o Car2 e Car1 após travagem	56
6.8	Velocidades do Car1 e Car2 a 60 km/h	57
6.9	Velocidades do Car1 e Car2 a 70 km/h	57

Lista de Tabelas

4.1	Descrição do Cenário	38
6.1	Parâmetros - Cenário Base	52
6.2	Parâmetros - Cenário para diferentes velocidades	55
6.3	Parâmetros - Cenários para diferentes frequências	58

Lista de Acrónimos

API	<i>Application Programming Interface</i>
BSP	<i>Basic Service Profile</i>
BSS	<i>Basic Service Set</i>
BSSID	<i>Basic Service Set Identifier</i>
BTP	<i>Basic Transport Protocol</i>
C-ITS	<i>Cooperative Intelligent Transport System</i>
CAM	<i>Cooperative Awareness Messages</i>
CSMA/CA	<i>Carrier Sensing Multiple Access with Collision Avoidance</i>
DCC	<i>Decentralized Congestion Control</i>
DENM	<i>Decentralized Environmental Notification Messages</i>
DSRC	<i>Dedicated Short-Range Communication</i>
ETSI	<i>European Telecommunications Standards Institute</i>
EUA	Estados Unidos da América
I2V	<i>Infrastructure to Vehicle</i>
ITS	<i>Intelligent Transport System</i>
MANET	<i>Mobile Ad-Hoc Network</i>
OBU	<i>On Board Unit</i>
ODE	<i>Open Dynamics Engine</i>
OMNeT++	<i>Objective Modular Network Testbed in C++</i>
OSI	<i>Open System Interconnection</i>
ROS	<i>Robotic Operation System</i>
RSU	<i>Road Side Unit</i>

SSID	<i>Service Set Identifier</i>
STE	Sistema de Travagem de Emergência
TCP	<i>Transmission Control Protocols</i>
UDP	<i>User Datagram Protocol</i>
V2I	<i>Vehicle to Infrastructure</i>
V2V	<i>Vehicle to Vehicle</i>
V2X	<i>Vehicle to Everything</i>
VANET	<i>Vehicular Ad-Hoc Network</i>
WAVE	<i>Wireless Access in Vehicular Environments</i>
WBSS	<i>Wireless Basic Service Set</i>

Capítulo 1

Introdução

1.1 Motivação

Na sociedade atual, cada vez mais a segurança e o conforto se tornam fundamentais no dia-a-dia. A existência de infraestruturas inteligentes permite um complemento que no futuro se pode tornar indispensável nas estradas e cidades de todo o mundo. Para além disso, os veículos cada vez mais têm a capacidade para receber informação e usá-la de forma a permitir uma maior assertividade nas decisões. Essa informação pode ser fornecida a partir de comunicações intra-veiculares (sensores do veículo) e extra-veiculares (infraestruturas colocadas de forma estratégica na via).

Com isto, foi então criado o conceito de *Cooperative Intelligent Transport System* (C-ITS) que se baseia na comunicação veicular para melhoramento da eficiência, relativa ao tráfego, e da segurança rodoviária. Essa partilha de informação é um recurso importantíssimo para melhorar o desempenho e garantir a segurança em aplicações *Safety-Critical*.

As comunicações *Vehicle to Vehicle* (V2V) e *Vehicle to Infrastructure* (V2I), são exemplos de tecnologias que permitem trocar o maior número de informação possível de forma a que haja uma maior segurança e estabilidade do veículo. Para isto se realizar, terão que ser feitos diversos testes e instaladas infraestruturas em cidades e estradas espalhadas por todo o mundo, garantindo a possibilidade de comunicação constante e perceção do ambiente que irá permitir a total automatização do veículo.

A prototipagem destes sistemas, na área automóvel, leva ao uso de vários recursos dispendiosos e de difícil acesso, que podem ser contornados, numa fase inicial,

recorrendo a plataformas de simulação que irão fornecer resultados precisos e simular todo o tipo de ambientes necessários para confirmação desses mesmos resultados.

Este estudo tem sido muito aprofundado nos tempos mais recentes pois é um dos aspetos fundamentais e um dos alicerces que dará vida aos veículos autónomos.

Neste trabalho, vai ser abordado o uso do protocolo 802.11p na comunicação V2V, V2I e *Infrastructure to Vehicle* (I2V), de forma a realizar uma *Safety-Critical application* que cumpra todas as normas de segurança de forma a poder ser aplicada futuramente em todo o tipo de sistemas.

Neste caso, será usada uma ferramenta que permitirá simular um ambiente de *cooperative platooning* que se baseia na cooperação entre veículos, em linha, numa determinada rota, que permite um melhor aproveitamento do combustível assim com uma maior segurança num ambiente real [14].

1.2 Contextualização

Este projeto e os seus resultados enquadram-se no âmbito do *FLOYD - "5G / SDN Intelligent Systems For low latency V2X Communications in cross-Domain Mobility Applications"*, Projeto de Investigação Colaborativa de Grande Escala CMU-Portugal (AAC n.º 04 / SI / 2019, Subsídio n.º 045912) liderado pela *Capgemini Engineering* e tendo como parceiros a *Altice Labs*, o Instituto de Telecomunicações, *Vortex CoLab*, o Centro de Pesquisa CISTER / ISEP e a *Carnegie Mellon University*. O objetivo do *FLOYD* é construir uma pilha de tecnologias para oferecer serviços de computação/rede de alto desempenho para utilizadores de veículos. Apesar do previsível surgimento de veículos conectados nos próximos anos, fornecer conectividade contínua e serviços de computação para clientes veiculares ainda é uma tarefa desafiadora, exigindo um trabalho coordenado entre vários participantes e a integração das respectivas tecnologias. O caso de uso básico do *FLOYD* é o de um pelotão de veículos que pode solicitar uma ampla variedade de recursos da infraestrutura para suportar aplicações de veículos (por exemplo, direção autónoma, navegação, infoentretenimento, controlo do pelotão, entre outros). O trabalho e os resultados deste projeto serão integrados na contribuição do CISTER / ISEP para o *FLOYD*, nomeadamente como uma componente de um ecossistema crescente de tecnologias e soluções orientadas para a condução autónoma conectada (e especificamente o *platooning*) que estão a ser desenvolvidos no CISTER / ISEP.

1.3 Objetivos e Descrição do Projeto

O objetivo principal deste projeto será realizar um Sistema de Travagem de Emergência (STE), de forma a evitar uma colisão entre pelotões de veículos, que será desenvolvido e avaliado em tecnologias de simulação como o Artery, o Gazebo e o

Objective Modular Network Testbed in C++ (OMNeT++). O desempenho das comunicações 802.11p em cenários de *Cooperative Platooning* serão também avaliados, com o objetivo de possibilitar a maior segurança possível na posterior implementação física. Consecutivamente, será também feita uma validação que estará presente no capítulo 6 relativamente ao *Cooperative Platooning* e ao STE num cenário de interseção perpendicular entre pelotões de veículos.

1.4 Calendarização

De forma a haver uma maior organização e controlo sobre os objetivos propostos, o projeto ficou distribuído da seguinte forma:

		May					June				July				Aug					S.
		3	10	17	24	31	7	14	21	28	5	12	19	26	2	9	16	23	30	6
Setup technologies																				
Implement RSU in OMNET																				
Finish scenario & evaluation																				
Write report																				
Revise report																				

Figura 1.1: Planeamento de Projeto

1.5 Organização do Relatório

Relativamente à estrutura do relatório, o capítulo 2 irá abordar as tecnologias de comunicação utilizadas para o desenvolvimento deste projeto como a arquitetura C-ITS, o protocolo IEEE 802.11p, assim como a norma mais bem aceite no panorama europeu (ETSI ITS-G5).

Já no capítulo 3, todos os pormenores pertencentes à simulação irão ser apresentados, como os simuladores utilizados, as suas *frameworks* e os pontos mais fortes que levaram ao seu uso. Para além disso, o *setup* de simulação irá ser descrito, destacando a interação entre simuladores.

Uma análise sobre o *cooperative platooning* estará presente no capítulo 4, assim como uma descrição pormenorizada de todo o cenário em que se baseou o caso de uso desenvolvido (STE). Para além disso, uma explicação da aplicação real do STE será apresentada de forma a perceber melhor como poderá ser feita uma aplicação física futura. Ainda neste capítulo, será apresentada uma tabela que apresenta todos os factos relativos a este mesmo caso de uso.

No capítulo 5, irá ser exposto todo o caso de uso desenvolvido neste projeto, explicando todas as suas partes e todos os seus processos, analisando grande parte do código desenvolvido na criação deste sistema.

A validação e resultados irá ser retratada no capítulo 6, com algumas conclusões e pontos a salientar a serem feitas sobre a simulação.

Por fim, as conclusões relativas ao projeto serão apresentadas, fazendo um resumo do desenvolvimento total e de algumas das dificuldades que foram surgindo com a evolução do trabalho. Para além disso, algumas palavras sobre o possível trabalho futuro como a implementação física serão apontadas.

Capítulo 2

Comunicações Veiculares - Tecnologias

Neste capítulo será dado um ponto de situação sobre o desenvolvimento das comunicações veiculares aplicadas em vários cenários de C-ITS [15], assim como também será explicada a forma como as mesmas podem ajudar na autonomia do veículo. A estrutura e arquitetura do protocolo 802.11p será abordada, passando pelas suas características e pelas suas vantagens relativas a outras possíveis tecnologias de comunicação veicular.

Para as aplicações apresentadas, a uso de um *standard* torna-se de especial relevância. Posto isto, será apresentado um dos *standards* mais utilizados nestas aplicações, o *European Telecommunications Standards Institute* (ETSI) *Intelligent Transport System* (ITS)-G5, com particular especificação para o caso do STE no qual se baseia o desenvolvimento do projeto.

2.1 Arquitetura C-ITS

O C-ITS é um paradigma de partilha de informações por parte dos veículos e das infraestruturas inteligentes que cada vez tem ganho maior relevância, resultando num maior desenvolvimento dos mesmos. Esta comunicação e informação faz parte do que se designa por *Vehicular Ad-Hoc Network* (VANET)[16]. Neste tipo de redes, a maior parte dos seus nós são móveis e capazes de tomar decisões sozinhos, sendo

por isso uma forma de *Mobile Ad-Hoc Network* (MANET) [17]. Na figura 2.1 está representado um típico cenário VANET.

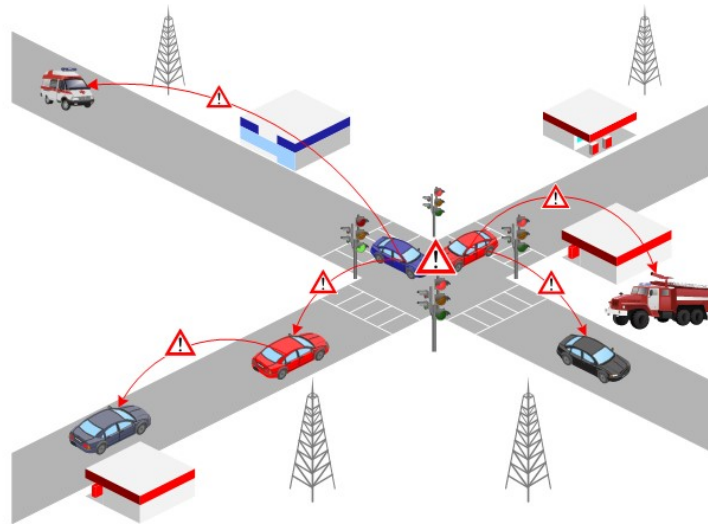


Figura 2.1: Cenário - VANET [1]

Pertencentes a esta rede veicular estão os principais componentes utilizados neste projeto como:

- **A *Road Side Unit* (RSU)** - Fixas, funcionam como unidades de comunicação que fornecem informação aos veículos num determinado raio de ação. Usadas como fontes de informação sobre as condições de tráfego;
- **A *On Board Unit* (OBU)** - São responsáveis por gerir as comunicações externas como as comunicações V2I e V2V.

Relativamente à estrutura C-ITS, os tipos de ligação I2V, V2I e V2V são de elevada importância neste projeto, podendo ser complementados por outros tipos de ligação (p.ex. satélite), representados na figura 2.2.

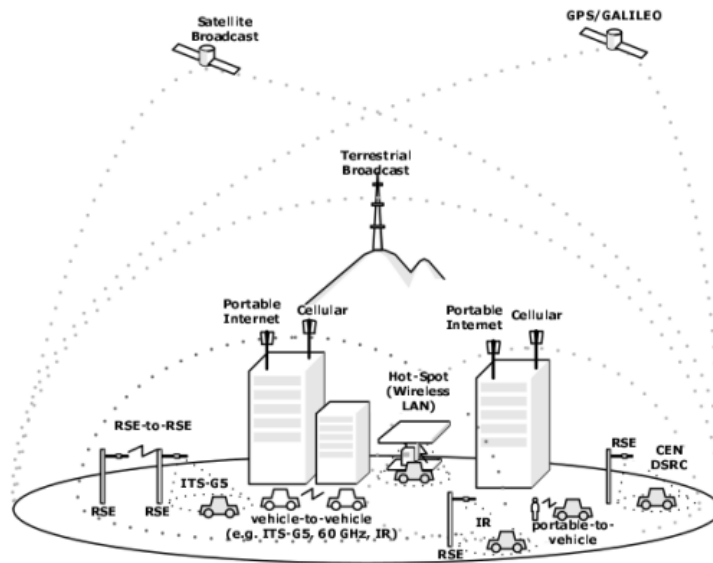


Figura 2.2: C-ITS - Tipos de ligação numa rede VANET [2]

Por fim, a conexão à *Internet* permitirá serviços que irão complementar a automação dos veículos, sem esquecer a necessidade de cumprir todas as condições de segurança.

2.2 Visão Geral das Tecnologias WAVE e ITS-G5

As tecnologias *Wireless Access in Vehicular Environments* (WAVE) e ITS-G5, relacionam-se como duas pilhas protocolares que utilizam o IEEE 802.11p nas suas camadas MAC e físicas. O WAVE diferencia-se por ser utilizado nos Estados Unidos da América (EUA) enquanto que o ITS-G5 é utilizado na Europa.

Relativamente ao IEEE 802.11p, o mesmo retrata uma especificação técnica do protocolo 802.11 em ambientes veiculares e, esta especificação do protocolo é distinta da utilizada na Europa, designando-se como *Dedicated Short-Range Communication* (DSRC) nos EUA. Como pode ser visto na figura 2.3, o WAVE e o ITS-G5 são empilhados no topo do IEEE 802.11p.

<i>OSI layers</i>	WAVE	ETSI TC ITS architecture		
<i>Application</i>	SAE BSM	CAM	DENM	<i>Facilities</i>
<i>Transport</i>	IEEE 1609.3	BTP		<i>Networking & transport</i>
<i>Network</i>		GeoNet		
<i>Data link</i>	LLC			<i>Access</i>
	IEEE1609.4	DCC		
	IEEE 802.11p			
<i>Physical</i>	IEEE 802.11p			

Figura 2.3: WAVE e ITS-G5 - Comparação [3]

Para além das várias parecenças entre as duas pilhas protocolares, as principais diferenças situam-se nas camadas de aplicação, de rede e de transporte, tal como é visualizado na figura 2.3, existindo também uma diferente organização das camadas. Apesar disso, as duas pilhas protocolares têm o mesmo objetivo, sendo usadas para todo o tipo de comunicações veiculares.

2.3 Protocolo IEEE 802.11p

O protocolo IEEE 802.11p[18] possibilita as comunicações V2V, I2V e V2I [19]. Desta forma, o IEEE 802.11p designa-se como um conjunto de protocolos que permitem garantir a comunicação entre veículos e infraestruturas, levando a uma maior fluidez por parte do tráfego.

Para conseguir atingir os seus objetivos, esta norma tem um leque de especificações regionais disponíveis para usar num ambiente veicular [20], ou seja, num ambiente onde existem mudanças muito rapidamente.

Relativamente à sua estrutura, o IEEE 802.11p é um ajuste na arquitetura IEEE 802.11 [21]. É baseado nos protocolos do IEEE 802.11a e tem as principais características do 802.11q.[22] As principais diferenças encontram-se na camada *MAC*, que teve que ser alterada para tornar as comunicações entre os veículos mais rápida e eficiente assim como também proporcionar um tempo de comunicação mais rápido devido às altas velocidades dos veículos [23]. Para além disso, o protocolo 802.11p estará também responsável pela camada física, como pode ser visto na figura 2.4.

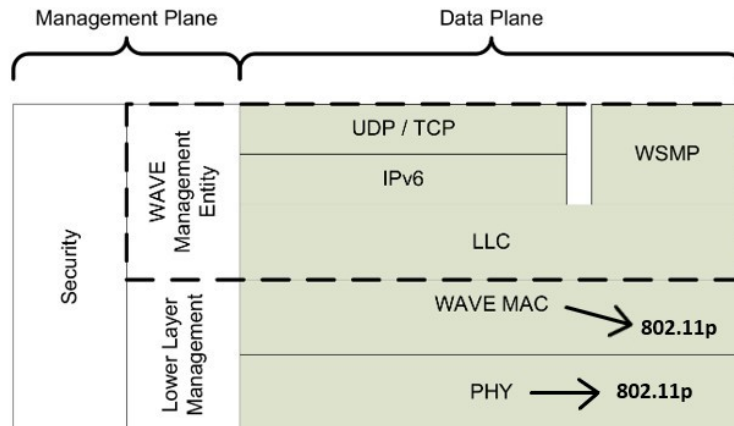


Figura 2.4: Exemplo - Arquitetura WAVE (EUA)

2.3.1 Camada MAC

De forma a enumerar as características do protocolo e perceber de que maneira contribuíram para as comunicações veiculares, existem 3 conceitos que necessitam de ser abordados como é o caso:

- **Basic Service Set (BSS)** - É uma parte importante do *standard* 802.11 e consiste num conjunto de estações que conseguem comunicar entre si;
- **Service Set Identifier (SSID)** - Representa o nome pelo qual a *network* se distingue e comunica, por exemplo, através de um *beacon frame* por um *access point*;
- **Basic Service Set Identifier (BSSID)** - Representa o *MAC adress* do *access point* do caso anterior.

Uma das características adicionadas no protocolo 802.11p, uma estação pode transmitir e receber *frames* com uma liberdade relativa ao *Wireless Basic Service Set* (WBSS) através do BSSID, pois os veículos partilham o mesmo, diminuindo o *overhead* a que estão sujeitos. Quando é enviado um WAVE *beacon* a partir deste mesmo modo, é partilhada informação do tipo de serviço e dos parâmetros necessários para a auto-configuração, fazendo com que se juntem mais veículos à WBSS. As estações utilizam este tipo de comunicação pois trata-se de uma forma segura de comunicação, sendo isso essencial nesta troca de informações entre veículos e infraestrutura. [24]

Ainda relativamente à camada *MAC*, é adotado pelo protocolo 802.11p o *Carrier Sensing Multiple Access with Collision Avoidance* (CSMA/CA), em que se tenta evitar que existam colisões investigando se o meio está livre para transmitir qualquer informação. Outro dos mecanismos utilizados é o mecanismo de *backoff* aleatório,

que permite enviar a mensagem depois de esperar um tempo aleatório para o fazer. Em contrapartida, estes mecanismos introduzem um *overhead* significativo[25].

2.3.2 Camada Física

Nesta camada, o IEEE 802.11p utiliza o IEEE 802.11a, fazendo alterações ao mesmo. Para a adequação do IEEE 802.11p aos ITS, existem diferenças que serão significativas e fundamentais. Essas diferenças serão ao nível das especificações do transmissor e recetor, sendo que os mesmos são mais severos assim como têm umas condições de rejeição de canais mais rigorosas [26]. Estas características são essenciais pois permitem diminuir a interferência na transmissão de informação, visto que irá ser criada uma quando uma fonte de interferência estiver perto do canal de destino ou quando o transmissor e o recetor estão afastados 10x mais do que a fonte e o recetor [13].

2.3.3 Vantagens

Por fim, o protocolo IEEE 802.11p, relativamente a outras tecnologias semelhantes como por exemplo o LTE-V2V, reflete-se como mais apropriado para situações onde existe uma grande movimentação de veículos, permitindo uma boa receção dos pacotes, com um atraso também diminuto [27]. Com estas particularidades, revela-se como um protocolo que poderá ser ainda mais desenvolvido e aproveitado no futuro para comunicações intra e extra-veiculares, satisfazendo o uso da sua aplicação neste projeto.

2.4 ETSI ITS-G5

Responsável por todo o tipo de comunicações apresentadas neste projeto, o ETSI ITS-G5 [28] é o *standard Vehicle to Everything* (V2X) mais aceite pelas indústrias automóveis na zona europeia, visto que nos Estados Unidos é mais aceite o IEEE 1609 WAVE [29], que será bastante semelhante ao próprio ETSI ITS-G5 em termos técnicos. Os dois utilizam o IEEE 802.11p para comunicação entre camadas subjacentes.

A alta complexidade dos sistemas ITS resulta numa grande variedade dos mesmos, com várias implementações e topologias que permitem por exemplo ao veículo receber informação importante sobre o estado da via, do tráfego e de toda a vizinhança a partir da RSU. Para além disso, neste projeto também estará incluída a comunicação V2V que pode ser descrita como na figura 2.5.

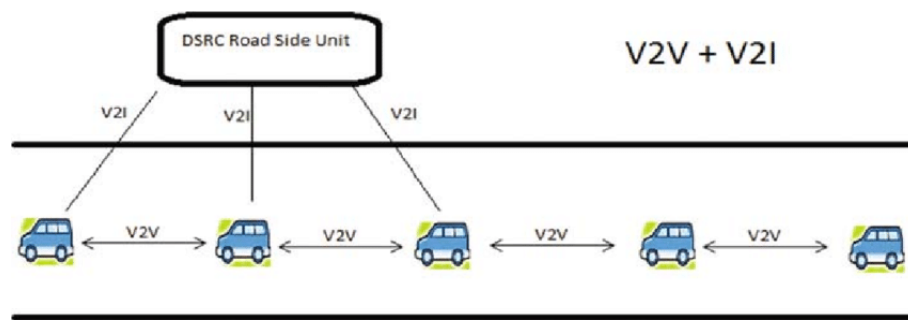


Figura 2.5: Rede de Sistemas ITS

Relativamente à estrutura do ETSI ITS-G5, na sua constituição existem 6 grandes grupos:

- **Applications** → Aplicação dos cenários ITS, se são usados para segurança, eficiência de tráfego, etc.;
- **Security** → Responsável pelas funcionalidades de segurança implementadas nesta norma;
- **Networking e transport** → Contém as funcionalidades e os protocolos dedicados às comunicações ITS;
- **Acesso** → A camada de acesso fornece os meios para aceder ao meio de comunicação;
- **Management** → Encarregado pelo bom funcionamento de todas as outras camadas;
- **Facilities** → Responsável pela troca de informação entre veículos e outras estações ITS.

A sua distribuição de forma mais compacta e funcional possível verifica-se na figura 2.6.

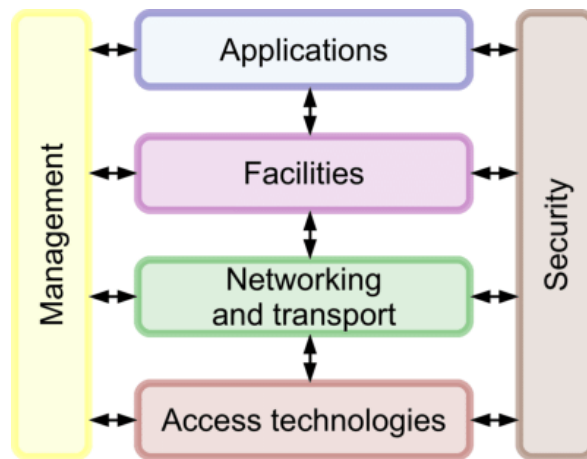


Figura 2.6: Distribuição - Estrutura ETSI [4]

Neste projeto, a principal zona de ação será na camada *Applications* e na camada *Facilities*, com a manipulação das *Decentralized Environmental Notification Messages* (DENM) e a troca de informações realizada pelos veículos e pela RSU. Agora, de forma a aumentar o nível de compreensão desta norma, irá ser abordada cada camada.

2.4.1 Applications

Como já foi esclarecido, existem vários tipos de cenários de aplicação, mas existe uma especial abordagem a:

- Cenários de eficiência de tráfego;
- Cenários de segurança na estrada.

Neste projeto, ambos os cenários estarão em evidência pois o evitar de colisão será referente à segurança na estrada mas irá existir também o *platooning* que se refere ao grupo de aplicação de eficiência de tráfego.

Para existir segurança, assim como rigor na receção e envio de informação a comunicação tem que ter condições que serão necessárias e obrigatórias para ter um cenário funcional e bem sucedido. Essas condições serão:

- Segurança;
- Fiabilidade;
- Latência.

Devido à necessidade de garantir estas condições, o ETSI criou com a ajuda de outros utilizadores o denominado *Basic Set of Applications*, no qual, para cada

possível caso de uso estão explícitos os requisitos necessários. Assim como os requisitos, também estão alguns dos exemplos mais relevantes de cenários para possível utilização.

CAM Messages

Estas mensagens são utilizadas de forma a atualizar as informações de localização de uma estação ou do estado da mesma. São enviadas, periodicamente por uma estação, para todas as outras que se localizam dentro da zona de comunicação especificada a partir do *Basic Set of Applications* [5].

Como a atualização de posição é um dos principais objetivos do envio deste tipo de mensagens, então o seu tempo de envio tem que obedecer a determinados requisitos que fazem com que não haja nenhuma falha de informação e com que tudo seja atualizado rapidamente. Esses requisitos, especificados no seu *Basic Service Profile* (BSP) estruturados pelo ETSI tem como principais fatores:

- Geração mínima de mensagem de 0.1 segundos;
- Geração máxima de mensagem de 1 segundo;
- Depois de ultrapassados determinados valores (velocidade, ângulo, distância), são enviadas mensagens automaticamente.

A frequência mínima de envio irá variar, podendo ser mais pequena ou maior, de acordo com qual o cenário utilizado, estando isso também evidente nos requisitos do ETSI.

Sendo assim, o envio de *Cooperative Awareness Messages* (CAM) pode ser estruturado como visualizado na figura 2.7.

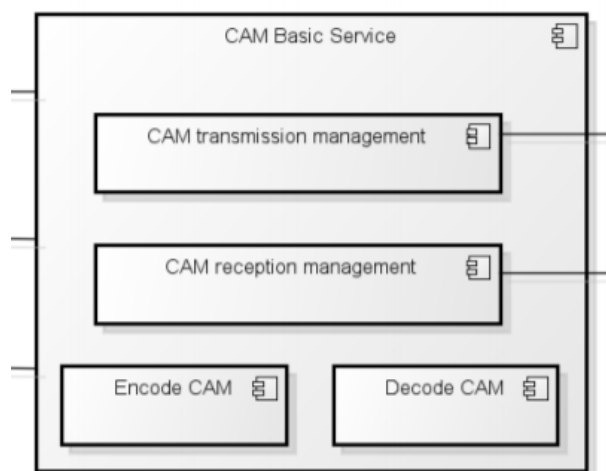


Figura 2.7: CAM - CaService [5]

Como é referido na figura 2.7, as mensagens são codificadas de acordo com a notação ASN.1 [30], sendo por isso necessário na estação de destino existir um decodificador para poder obter as informações.

Por fim, o ETSI permite a existência de diferentes tipos de estações ITS que criarão diferentes hierarquias de prioridade no envio de CAM como:

- **BasicVehicle** → Os veículos privados, servindo como base para os restantes;
- **basicIRS** → Uma *RoadSide Station* que funciona como uma infraestrutura;
- **EmergencyVehicle** → Que como o nome indica, designa um veículo de emergência como ambulância, bombeiros, etc.;
- **publicTransportVehicle** → Que se destina aos transportes públicos.

DENM Messages

As DENM são mensagens que são desencadeadas a partir de eventos. Utilizadas por todas as estações, funcionam como um aviso de um evento desencadeado numa certa área geográfica.

Antes de serem esclarecidos os passos de deteção, no envio de DENM podem ser especificados 4 *containers*. Cada um deles fornecerá informação aquando do envio da mensagem. Deste modo, eles são:

- *Management container* → De uso obrigatório, contém informação sobre a gestão da DENM e sobre os seus protocolos;
- *Situation container* → Contém informação relativa ao tipo de evento detetado;
- *Location container* → Contém informação sobre a localização do evento;
- *À la carte container* → Contém informação adicional (se necessária) sobre o caso de uso utilizado.

De forma a explicar melhor o desencadeamento das DENM, serão então enumerados os passos de deteção e algumas características.

1. Quando detetado o evento pela estação ITS, é enviada uma DENM para todas as estações presentes numa determinada área geográfica;
2. A DENM é enviada enquanto o evento estiver a acontecer, sendo repetida compulsivamente a uma determinada frequência;
3. A terminação da DENM é acionada quando o evento desaparecer de forma automática ou quando uma estação ITS estabelecer o fim do evento;

4. A informação das DENM é usada pelas estações a qual a informação interessa, sendo elas que decidem o que fazer.

A estruturação da DENM é feita no *DenService* e está apresentada na figura 2.8, com os seus passos mais importantes.

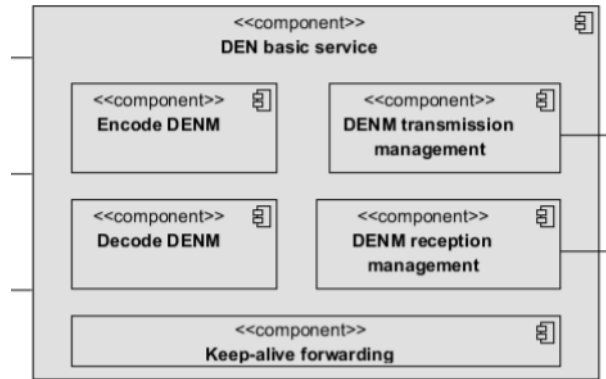


Figura 2.8: DENM - DenService [6]

Toda a informação proveniente da DENM é utilizada pela camada *Facilities* do ETSI ITS-G5 para atualização do *Local Dynamic Map*.

Por fim, como mostra a figura 2.8, é também usada a codificação por parte da notação ASN.1.

2.4.2 Security

Relativamente à segurança, a norma ITS-G5 tem implementada uma série de funcionalidades que permitem manter uma grande segurança e fiabilidade em todos os cenários. Essas funcionalidades estarão demonstradas na figura 2.9.

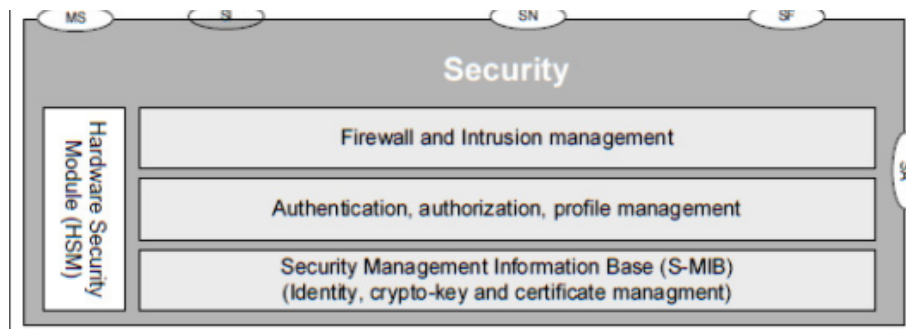


Figura 2.9: Security - Funcionalidades [2]

2.4.3 Networking e Transport

Com funcionalidades das camadas de rede e transporte *Open System Interconnection* (OSI), esta camada diferencia-se pela existência de algumas implementações destinadas às comunicações ITS como:

- Protocolos de rede;
- Protocolos de transporte;
- Interface para todas as outras camadas de comunicação ITS;
- Gestão das camadas de transporte e de rede.

Relativamente aos protocolos de rede, o *GeoNetworking* controla o transporte dos pacotes de informação desde que sai da origem até ao seu destino, esse destino que pode ser:

- Um nó individual (GeoUnicast);
- Todos os nós presentes numa área geográfica (GeoBroadcast);
- Todos os nós de uma determinada vizinhança utilizando o *single-hop broadcast*.

Para uso nestes cenários de segurança e eficiência de tráfico, o *single-hop broadcast* é utilizado para o envio periódico das CAM [31] enquanto que para as DENM é utilizado o *GeoBroadcast* para enviar a informação para uma determinada área.

Para além disso, o *GeoNetworking* utiliza três algoritmos para melhorar a sua *performance* como é o caso do *Single Geo-Broadcast*, o *Advanced Forwarding* e o *Contention-Based Forwarding* [32].

Retornando então aos restantes protocolos de rede, o IPv6 *networking* é também suportado pela norma ITS-G5 assim como o IPv4 para transição para IPv6.

Já relativamente aos protocolos de transporte, são suportados 3 tipos, sendo eles:

- *Transmission Control Protocols* (TCP);
- *Basic Transport Protocol* (BTP);
- *User Datagram Protocol* (UDP).

Os protocolos UDP e TCP, largamente conhecidos, neste caso, não são os mais usados (apesar de também largamente utilizados), sendo superados pelo protocolo BTP. Este é manuseado sobre o protocolo de *GeoNetworking*, mas a integração do TCP e UDP também é possível.

Com isto, resumidamente, a camada *Networking e Transport* está estruturada de acordo com a figura 2.10, estando evidentes os seus pontos mais importantes.

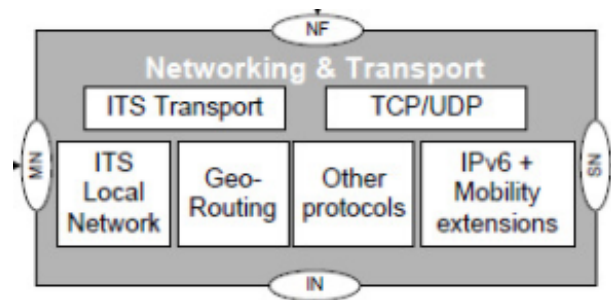


Figura 2.10: Networking e Transporte - Principais pontos [2]

2.4.4 Camada de Acesso

A grande necessidade de comunicações constantes e a partilha de grandes dados de informação leva a que possa haver congestão e com que seja excedida a capacidade dos canais de transmissão. Para isto, foi implementado então um sistema que permite regular as comunicações de forma a não exceder o limite evitando o mau funcionamento do sistema. Esse sistema chama-se *Decentralized Congestion Control* (DCC). Deste modo, a qualidade de serviço é garantida, assim como controla a entrega de mensagem no momento correto sem nenhum tipo de obstrução no caminho [33].

2.4.5 Management

Esta camada, responsável pela gestão metódica de todas as outras camadas, tem o objetivo de garantir todo o bom funcionamento do sistema. Para além disso, tem que manter a ordem, promovendo a cooperação constante entre camadas.

Na figura 2.11, são apresentados os parâmetros de gestão das camadas:

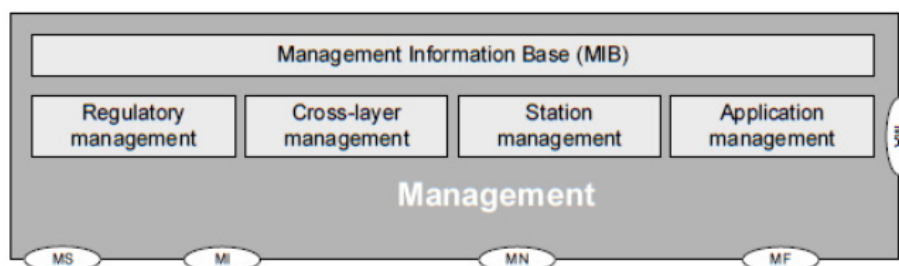


Figura 2.11: Management - parâmetros [2]

2.4.6 Facilities

Esta camada designa-se como a camada mais importante de todo este sistema, permite gerir toda a informação proporcionada por todas as estações ITS, quer os veículos quer a RSU. O uso de algumas funcionalidades presentes na camada OSI, torna esta camada bastante complexa. Funcionalidades da camada de apresentação,

da camada de aplicação e da camada de sessão estão presentes, como visualizado na figura 2.12.

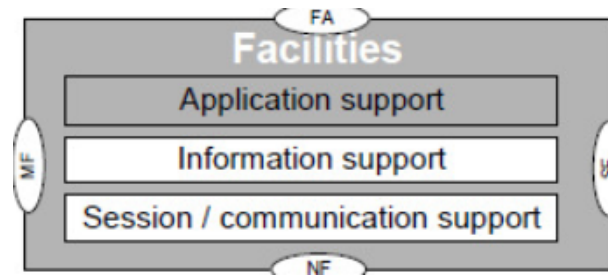


Figura 2.12: Facilities - Funcionalidades [2]

Desta maneira, estas são algumas características que a tornam essencial para este sistema, como:

- Fornecer uma estrutura que permite codificar e decodificar as mensagens, que como já vimos são codificadas de acordo com a notação ASN.1;
- Fornecer informação de localização geográfica e informação temporal;
- Fornecer uma gestão de troca de mensagens DENM e CAM entre estações ITS;
- Fornecer suporte de repetição de mensagens DENM ou CAM;
- Fornecer informações sobre a estrada e todos os objetos presentes na mesma, assim como também da possível passagem de peões.

Assim, denota-se a vasta informação que é fornecida por esta camada, ressaltando então a sua importância nestes sistemas.

Capítulo 3

Tecnologias de Simulação

A validação de todo o projeto foi realizada em plataformas de simulação. Neste caso, a junção de plataformas robóticas e de redes de comunicação serão as mais relevantes e permitiram criar uma cooperação de forma a atingir o objetivo final.

Neste capítulo, serão esclarecidas quais as plataformas utilizadas assim como também algumas das características das mesmas.

Seguidamente, será então também abordado o Artery, uma ferramenta de simulação que permite executar simulações com o protocolo ETSI ITS-G5 no OMNeT++, uma plataforma de rede responsável pela comunicação entre módulos.

Depois, na secção 3.4, será debatido o *Robotic Operation System* (ROS) e as características que irão relacionar o Gazebo, plataforma robótica que irá permitir uma representação física, ao OMNeT++, permitindo o controlo das duas plataformas em simultâneo, assim como a integração de todo o projeto.

Por fim, na secção 3.5 será descrita a ferramenta do COPADRIVe e o que a mesma fornece ao STE, acabando com a secção 3.6 que apresenta todo o *Setup* de simulação, relacionando todas estas ferramentas e simuladores.

3.1 OMNeT++

O OMNeT++ [34], um dos grandes simuladores utilizados neste projeto, corresponde a um simulador orientado a objetos. Dividido em módulos, o OMNeT++ tem como objetivo ser utilizado para meios académicos e para pesquisas. Derivado disso, uma

das grandes vantagens tem o seu uso poder ser feito sem custos adicionais, ou seja, ser *open source*.

A sua vasta aplicabilidade para projetos de simulação de redes de comunicação modulares, micro-processadores e outros sistemas paralelos, leva a que o mesmo seja cotado como um simulador de redes, mas, na realidade trata-se apenas de um simulador, sem especialização em redes [35]. Isto pois, todos os componentes por parte das redes de computadores são fornecidas pelo INET, uma das possíveis *frameworks* utilizadas no simulador, ou seja, totalmente independentes do OMNeT++ em si.

Relativamente à sua estrutura, a sua divisão em módulos permite que, se desenhados de forma correta possam ser aplicados em diferentes ambientes, destacando-se pela sua versatilidade. A sua comunicação é realizada por *message passing*, ou seja, é realizada uma comunicação entre processos. Já em relação à composição dos módulos, os mesmos são escritos em *C++* usando uma biblioteca de simulação. As mensagens, entre módulos, podem ser enviadas diretamente ao destinatário ou passadas por entre as conexões desses próprios módulos.

A figura 3.1 demonstra então a composição de uma *network* assim como a formação de módulos compostos através de módulos simples, dando então a ideia de um funcionamento estrutural.

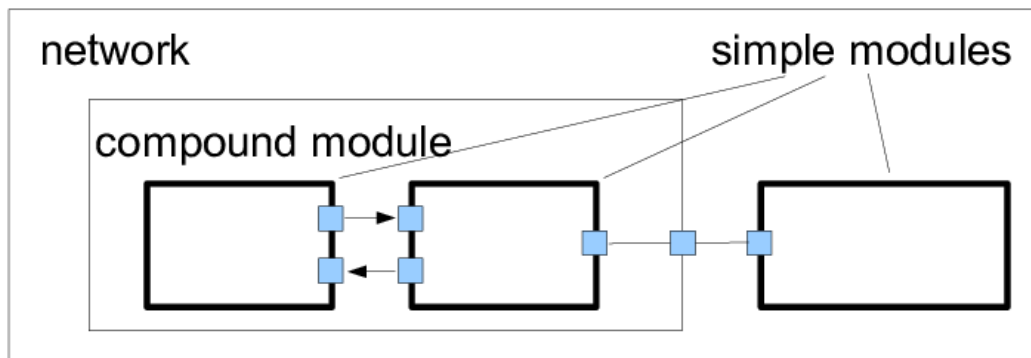


Figura 3.1: Estrutura por Módulos - OMNeT++ [2]

Para manipular e definir melhor a estrutura do modelo (módulos e as suas conexões), existe um ficheiro programável em linguagem NED, a linguagem de descrição do OMNeT++. Neste ficheiro, são apresentadas as conexões e parâmetros dos módulos simples e compostos, assim como também, nestes últimos, das definições dos sub-módulos e conexões internas. Para além disso, são definidas as *networks* que se exibem como modelos independentes da simulação.

Resumidamente, os ficheiros *C++* referem-se à aplicação do código e de ações, enquanto que os ficheiros *.ned* se referem à parte gráfica do modelo, definindo os parâmetros e conexões. Com isto, a existência de 3 ficheiros relativos a um módulo (*.cc*, *.h* e *.ned*) é usual nos modelos do OMNeT++.

Já para o cenário de simulação, existe outro ficheiro que trata de colocar valores iniciais, possíveis de alterar antes de correr a simulação de forma a avaliar os resultados. Esse ficheiro denomina-se *.ini file* e tem a função de guardar os valores para os quais a validação é efetuada.

Um dos pontos vantajosos do OMNeT++ é o seu interface gráfico, que permite que haja uma correlação entre a parte gráfica e a parte de código dentro do próprio OMNeT++, tornando o mesmo mais apelativo e mais fácil de manipular. Essa interface é apresentada na figura 3.2.

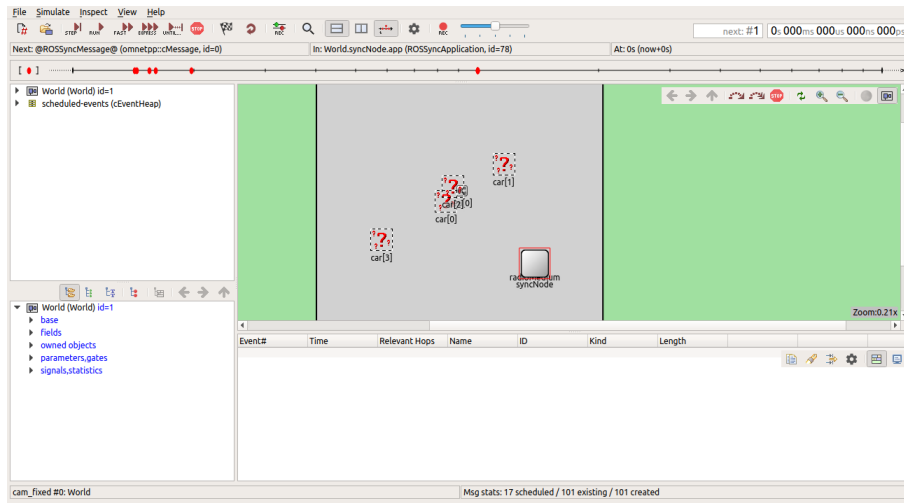


Figura 3.2: OMNeT++ - Interface Gráfico

Relativamente à simulação, existem 3 possíveis interfaces disponíveis no OMNeT++:

- **Qtenv**
- **Tkenv**
- **Cmdenv**

Neste caso, será utilizada a interface Qtenv que permite a integração com o *clock* do Gazebo ao selecionar a opção *"Fast"* ao iniciar a simulação. Esta foi também uma das razões para a escolha do OMNeT++ como simulador de redes deste projeto, assim como também a sua grande versatilidade e possibilidade de ser o *host* ao lado do Gazebo.

3.2 Artery

O Artery [36], destaca-se como um projeto que permite simulações V2X baseadas em protocolos ETSI ITS-G5, sendo também o primeiro a integrar um simulador de redes (OMNeT++) com um simulador de tráfego automóvel (SUMO). Começando como

uma extensão do *Veins*, o *Artery*, foi desenvolvido até ao ponto de se poder utilizar de forma totalmente independente do mesmo. Para além disso, a expansão para o uso do *standard* ETSI ITS-G5 é possível a partir do *Vanetza*, que se caracteriza como a única implementação *open source* dos protocolos ETSI C-ITS [37].

Relativamente à sua estrutura, o *Artery* reflete-se como uma "*camada*" implementada no topo do *Veins* [38] e do *Vanetza* [39] e define-se como uma *framework* versátil de implementação de serviços. Desta forma, a partir da figura 3.3 é de fácil perceção a relação entre estas 3 *frameworks*, apresentando uma visão geral das suas estruturas.

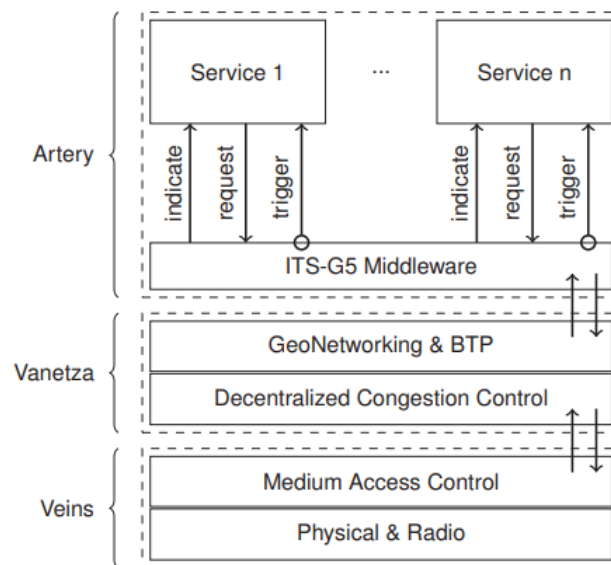


Figura 3.3: Estrutura - Artery [7]

Como se observa na figura, grande parte do *Artery* se baseia no *ITS-G5 Middleware* que tem como objetivo funcionar como uma espécie de base para cada serviço disponibilizado pelo *Artery*, fornecendo informações para todos os componentes assim como também faz uma inicialização dos mesmos.

Já entre o *Middleware* e os serviços existem 3 conceitos assinalados na figura que merecem especial atenção:

- **Trigger** → O seu funcionamento decide se a mensagem é enviada ou não. Isto é, se umas determinadas condições forem cumpridas a mensagem é enviada, podendo funcionar como uma forma de realizar *updates* de posição do veículo, da sua velocidade, etc.
- **Request** → No qual é realizado um pedido pelo serviço de forma a avisar as camadas inferiores da intenção de enviar uma mensagem. A mensagem só é transmitida quando a camada de acesso permitir, até lá, apenas fica em espera.

- **Indicate** → Indicação das camadas inferiores que a mensagem foi recebida e que é permitido aceder ao conteúdo da mesma, ou seja, aceder ao conteúdo da mensagem enviada por outro membro, como uma RSU ou um veículo.

Relativamente ao projeto, o Artery implementado no COPADRIve (e utilizado neste projeto), terá dois serviços, o CAM service denominado *CaService*, completamente funcional e o DENM service denominado *DenService*, que sofrerá as alterações necessárias como será abordado mais à frente.

3.3 Gazebo

Outro dos simuladores utilizados neste projeto, o Gazebo [40], destaca-se como um simulador robótico "open source" que permite uma renderização de alta qualidade, incluindo sombras, iluminação e texturas. Tem a capacidade de utilizar vários motores físicos de alto rendimento como por exemplo o *Open Dynamics Engine* (ODE) [41] e o *Bullet* [42].

Para além disso, é um simulador totalmente dependente do ROS, não podendo ser usado de outra forma. Essa dependência traz benefícios, como a possível sincronização com outro tipo de simuladores, como por exemplo o OMNeT++, de maneira a realizar uma simulação perfeitamente realista [43]. Este, para além de outras características apresentadas mais à frente, será um dos pontos principais para o uso do mesmo neste projeto.

Relativamente à sua programação e à sua interface, destaca-se pela sua simplicidade e fácil adaptação, como observado na figura 3.4.

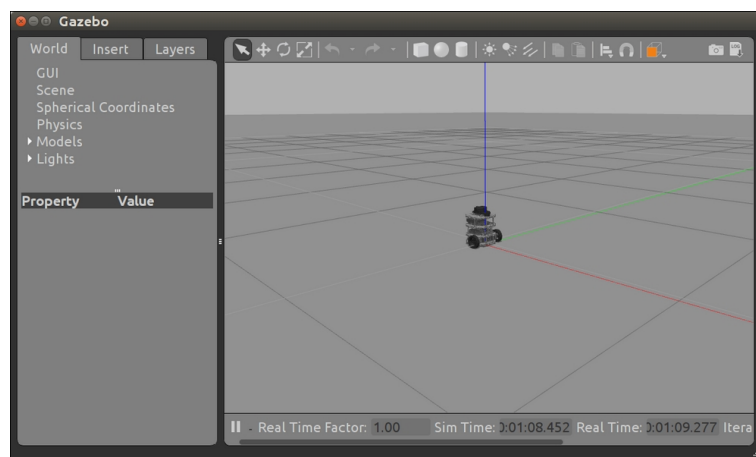


Figura 3.4: Interface Gazebo

Neste projeto, foi utilizado o Gazebo 9, devido a ser a versão também utilizada pelo COPADRIve, desta forma evita-se qualquer tipo de problemas de compatibilidade. Complementado ao Gazebo 9, estará a distribuição do ROS denominada por *Melodic* [44], que será a distribuição mais apropriada para usar no *Ubuntu 18.04*.

A possível implementação de diferentes cenários, devido a uma grande variedade de sensores, permite produzir simulações e ambientes bastante idênticos aos encontrados no dia-a-dia. Desta forma, será mais um dos pontos principais para a utilização de ferramentas de simulação para desenvolvimento inicial de projetos, dar uma visualização de um cenário e elementos reais, como podemos ver nas figuras 3.5 e 3.6.

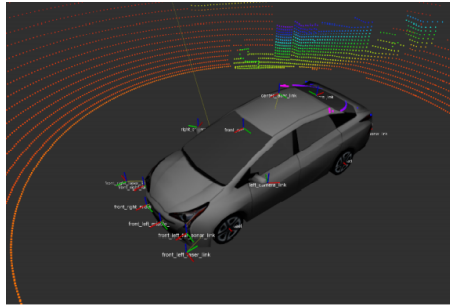


Figura 3.5: Veículo Utilizado

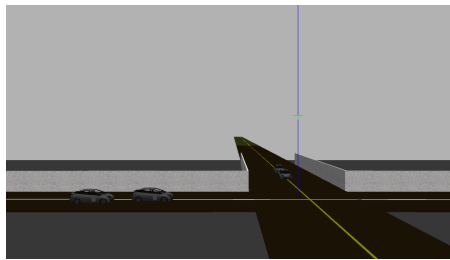


Figura 3.6: Cenário - Interseção

Para além disso, existem algumas características que fazem deste simulador um dos mais utilizados nesta área como por exemplo:

- Existência de *plugins* para controlar *robots* e ambientes, desenvolvidos pela *Application Programming Interface* (API) do Gazebo;
- Modelos de *robots* já desenvolvidos prontos a usar;
- Possível correr a simulação a partir de ferramentas da linha de comandos (Utilizado neste projeto).

Com isto, a utilização do Gazebo mostrou-se como a mais acertada devido à grande simplicidade de processos de acordo com as grandes potencialidades do próprio simulador que permitiram uma fluidez do trabalho e de tiragem de resultados assim como avaliação dos mesmos.

3.4 Robot Operation System (ROS)

Em forma de elo de ligação entre simuladores, o ROS [45], designa-se como uma *framework* robótica que fornece ferramentas e bibliotecas para desenvolvimento de

aplicações robóticas. Para além de ser usado por simuladores, também permite ser utilizado em projetos baseados em *hardware*, interligando os componentes do mesmo.

A possibilidade de fornecer "*message-passing*", ou seja, passagem de mensagem entre processos, assim como também o seu uso poder ser "*open source*", revelou ser um dos pontos fulcrais para a utilização desta *framework* no contexto do COPA-DRIVE e neste projeto.

De forma a perceber o funcionamento explicado a seguir, existem alguns conceitos que deverão ser esclarecidos como:

- **Nós** - Designam-se por nós os processos relativos ao projeto, sendo que um sistema terá vários processos que irão controlar o módulo no total;
- **Tópicos** - São os meios usados para comunicação entre nós, funcionamento com um sistema de Publicação/Subscrição;
- **Bags** - São uma forma de guardar e reproduzir informação de uma mensagem ROS.

O ROS, relativamente ao seu funcionamento, tem um conceito semelhante ao *Node.js* [46]. Esse conceito, baseia-se num sistema de Publicação/Subscrição, em que cada nó pode publicar e subscrever a qualquer tópico de forma a trocar informações com outro nó. Como podemos ver na figura 3.7, um nó publica uma mensagem para um determinado tópico e cada um dos nós subscritos a esse tópico recebem a mensagem.

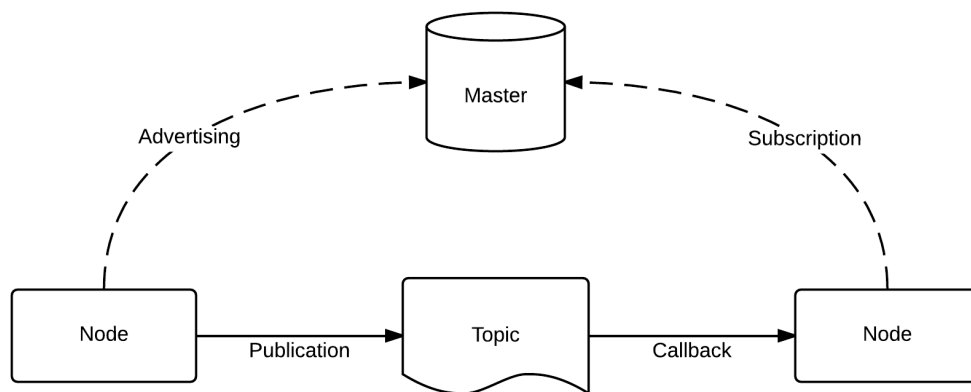


Figura 3.7: ROS - Publicação/Subscrição [8]

Para desenvolvimento e programação, o ROS suporta *C++* e *python* tendo também já sido realizados alguns testes em bibliotecas *java*.

Assim como o Gazebo, tem ferramentas que permite o seu uso pela linha de comandos assim como também uma parte mais gráfica, a partir do *rqt graph*, que

fornece uma visualização dos nós e dos tópicos ROS. Em forma de exemplo, a figura 3.8 reflete essa visualização de forma bastante clara, permitindo uma melhor percepção dos processos.

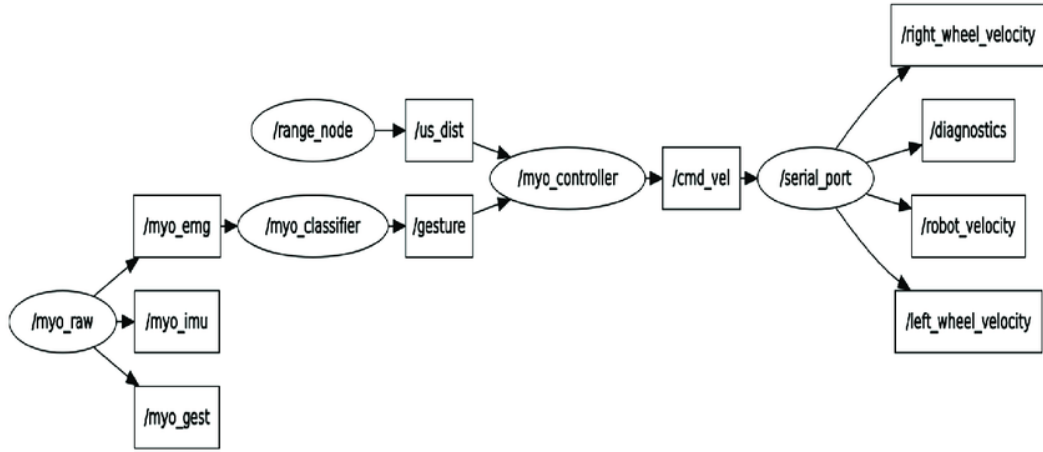


Figura 3.8: ROS - Exemplo [9]

Desta forma, todos os processos relativos ao projeto irão ter por base o sistema operativo ROS, revelando-se então como uma parte fundamental para o desenvolvimento do mesmo.

3.5 COPADRIVe

Por último, a ferramenta que auxilia o uso do *cooperative platooning* tem uma estrutura que pode ser definida em passos como os visualizados na figura 3.9.

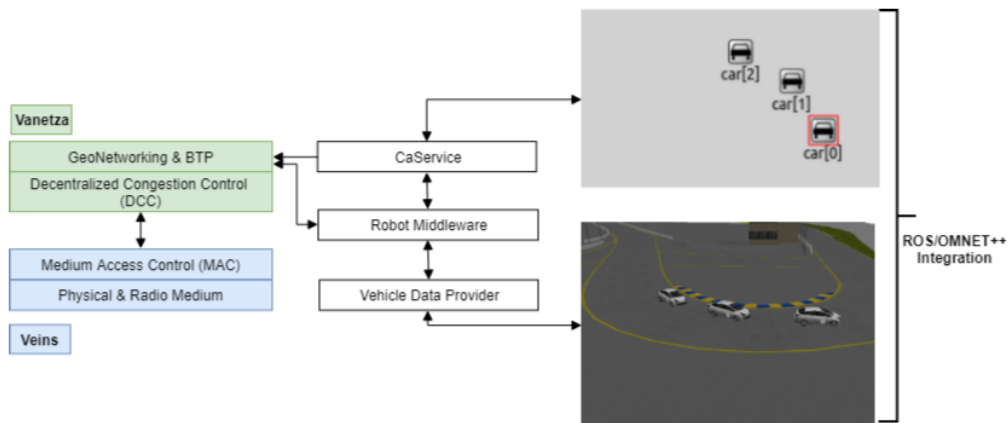


Figura 3.9: COPADRIVe - Estrutura [10]

Relativamente à estrutura principal do Artery, o COPADRIVe destaca-se como uma diferente implementação do *CaService*, do *Robot Middleware* e do *Vehicle Data*

Provider, assim como uma sincronização entre Gazebo e OMNeT++. Para além disso, utiliza externamente os protocolos de comunicação do *Vanetza* e do simulador *Veins*.

O módulo do *Vehicle Data Provider* refere-se a toda a informação adquirida pelos sensores do veículo e permite que os nós do OMNeT++ percebam como e onde está o veículo do Gazebo. Essa informação do veículo é fornecida a partir da subscrição feita em um dos tópicos ROS, denominado "*carX/carINFO*",

```
void VehicleDataProvider::RXNetCallback (const ros_msgs::OMNET_CAM &msg) {

    car_name = msg.car_name;

    ros_heading = msg.heading;

    ros_speed = msg.speed; // m/s

    ros_acceleration = msg.acceleration;

    double R = 6731; // km
    ros_x = msg.latitude;
    ros_y = msg.longitude;
    ros_z = msg.altitude;
    ros_latitude = (asin(ros_z/R) * 180.0 / M_PI); // -9000000000 9000000000 _ 0.1 microdeg
    ros_longitude = (atan2(ros_y,ros_x) * 180.0 / M_PI); // -1800000000 1800000000 _ 0.1 microdeg
```

Figura 3.10: Vehicle Data Provider - RXNetCallback()

Como demonstra a figura 3.10, nesta função é armazenada a informação proveniente do Gazebo, sendo que apenas o nome do carro, o seu ID assim como a sua localização são armazenados. Com a ajuda de um outro ficheiro de nome *Mobility-ROS*, a informação é atualizada no OMNeT++.

O *CaService* refere-se ao serviço utilizado para envio de CAMs entre veículos para atualização de posição e perceção de todos os movimentos do veículo. A sua taxa de envio de mensagem terá que ser alta, para o sistema ter sempre as posições atualizadas e poder ter tempo de resposta para que se algo de anormal aconteça, seja dada uma resposta de forma ágil. Essa taxa de envio, assim como quando será enviada a mensagem, é especificada no *CaService*. Para ajudar nessa especificação, o ETSI TS 102 637-2 [5] terá informação importante e fornecerá o seu BSP. A diferente implementação relativamente à fornecida pelo Artery, destaca-se pelo uso do *Vehicle Data Provider* para fornecer informação e colocar a mesma nos respetivos parâmetros CAM, como podemos ver na figura 3.11.

```

vanetza::asn1::Cam createCooperativeAwarenessMessage(const VehicleDataProvider& vdp, uint16_t genDeltaTime)
{
    vanetza::asn1::Cam message;
    //std::cout << "createCooperativeAwarenessMessage (1) " << std::endl; //DEBUG_ENIO
    ItsPduHeader_t& header = (*message).header;
    header.protocolVersion = 1;
    header.messageID = ItsPduHeader__messageID_cam;
    header.stationID = vdp.station_id();

    CoopAwareness_t& cam = (*message).cam;
    cam.generationDeltaTime = genDeltaTime * GenerationDeltaTime_oneMilliSec;
    BasicContainer_t& basic = cam.camParameters.basicContainer;
    HighFrequencyContainer_t& hfc = cam.camParameters.highFrequencyContainer;

    basic.stationType = StationType_passengerCar;
    basic.referencePosition.altitude.altitudeValue = AltitudeValue_unavailable;
    basic.referencePosition.altitude.altitudeConfidence = AltitudeConfidence_unavailable;

    if(abs(vdp.longitude().value())>Longitude_unavailable)
        basic.referencePosition.longitude = 0.0; //round(vdp.Latitude(), microdegree) * Latitude_oneMicrodegreeNorth;
    else
        basic.referencePosition.longitude = vdp.longitude().value(); //round(vdp.Longitude(), microdegree) * Longitude_oneMicrodegreeEast;
}

```

Figura 3.11: CaService - createCooperativeAwarenessMessage()

Já o *Robot Middleware* funciona como um elo de ligação entre os nós do ROS/Gazebo e os módulos do OMNeT++. É nele também que os veículos recebem as mensagens exteriores no ROS e onde são feitas as restantes inicializações dos módulos do OMNeT++. Neste caso, é responsável por armazenar a informação proveniente de outros veículos. Essa informação é enviada a partir da subscrição a um tópico ROS por parte do veículo no Gazebo. Na figura 3.12, é apresentada a função *ReceiveSignal()* que trata a informação quando ela é recebida por um outro veículo, havendo então a sua publicação no tópico ROS referente.

```

void RobotMiddleware::receiveSignal(cComponent* source, simsignal_t signal, cObject *obj, cObject*)
{
    //num_receive_msg_RM = num_receive_msg_RM +1;
    uint32_t real_vehicle = 1; //variable that defines if is a real vehicle and should record the message

    if ((signal == scSignalCamReceived) ) {
        numReceived_RM++;
        auto* cam = dynamic_cast<CaObject*>(obj);
        if (cam) {
            uint32_t stationID = cam->asn1()->header.stationID;

```

Figura 3.12: Robot Middleware - ReceiveSignal()

Para além destes ficheiros colocados de forma a estabelecer comunicação entre o Gazebo e OMNeT++, existe uma parte que deverá ser também considerada, a sincronização dos dois simuladores. Desta forma, foi criado um ficheiro para que quando fosse simulada toda a *framework*, tudo estivesse a acontecer ao mesmo tempo, tanto a nível de envio de mensagens como também de receção. Esse ficheiro, de seu nome *ROSSyncApplication* permite então sincronizar o *clock* dos dois simuladores, trabalhando como um só. Na figura 3.13 encontra-se uma porção dessa sincronização, neste caso do envio da mensagem.


```

void ROSSyncApplication::handleMessage(cMessage *msg) {
    // Handle time synchronization message
    if (msg == syncMsg) {
        // Sync with ROS
        unique_lock < std::mutex > lock(syncMutex);
        syncCondition.wait(lock);
        lock.unlock();

        // Schedule next sync invocation
        scheduleAt(ros::Time::now().toSec(), syncMsg);
    } else {
        cerr << "ROS sync application received unexpected message" << endl;
    }
}

```

Figura 3.13: ROSSyncApplication - handleMessage()

De forma a ter uma melhor percepção da conexão entre simuladores e do que é realizado por cada um deles, a figura 3.14 representa alguns dos processos de acordo com o tempo de simulação, ou seja, de forma cronológica da esquerda para a direita.

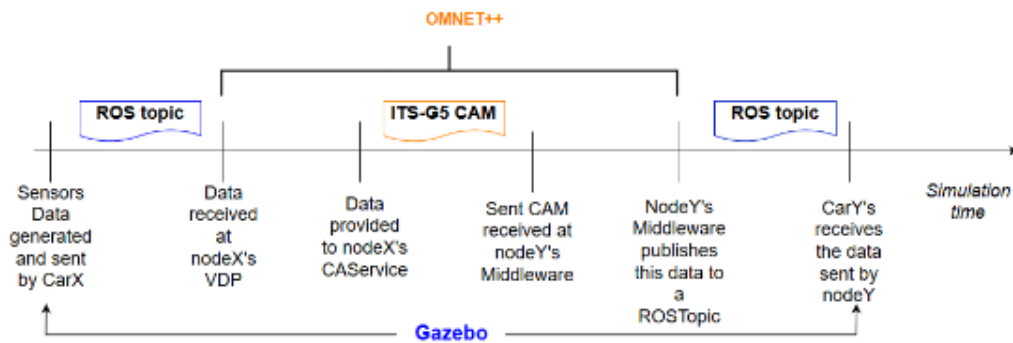


Figura 3.14: COPADRIVe - Estrutura [10]

De acordo com a figura, a informação dos sensores do "CarX" é enviada do Gazebo através dos vários nós do OMNeT++ para outro carro pertencente ao mundo do Gazebo, neste caso o "CarY", sendo este o conceito de funcionamento das CAM. Os designados por "NodeX" e "NodeY" serão as equivalentes *interfaces* da rede do "CarX" e "CarY", respetivamente. Com isto, estará perceptível a relação e a importância do ROS como comunicação a partir de tópicos ROS entre o Gazebo e o OMNeT++.

Ainda relativamente a ficheiros fornecidos pelo COPADRIVe, o *platooning.py*, ficheiro que permite que os veículos se movimentem em pelotão [47], será utilizado posteriormente na base de simulação servindo como acréscimo ao trabalho desenvolvido. Na figura 3.15 está representada uma porção do código, referente ao controlo de distância do pelotão.

```

def distance_control (d_error):
    #Global variables-----
    global dis_integral
    global dist_deriv
    global erro_dis_old

    global time_distance_control          #time of last distance control interaction
    global pid_adjust_old                 #in case of interactions smaller then dt

    debug = OFF                          #defines if the text will be printed
    #Calcule dt-----
    #used to determine the time between interactions from the program
    time_now = tp.time()                 #saves te time of the system
    dt = time_now - time_distance_control #time between interactions
    if (dt >= MINIMUM_DT):                #time between control actions
        #if (1):                          #time between control actions
            time_distance_control = time_now #save the time for next interaction

        #determines the integrator component
        if (abs(dis_integral) >= MAX_ERRO_INTEGRADOR_DIS or dt>10):
            dis_integral = ZERO
        else:
            dis_integral = dis_integral + d_error * dt

        #derivative value
        if (dt < 1):
            dist_deriv = (erro_dis_old - d_error)
        else:
            dist_deriv = (erro_dis_old - d_error)
        erro_dis_old = d_error

        #calcula PID de distancia
        PID_dis_adjust = kp_dis * d_error + ki_dis * dis_integral + kd_dis * dist_deriv
        pid_adjust_old = PID_dis_adjust
    else:
        PID_dis_adjust = pid_adjust_old
        print "Time interval < dt (DIST)"
    if debug:
        print ("PID_Dis_adjust: ", PID_dis_adjust)
    return PID_dis_adjust

```

Figura 3.15: platooning.py - Distance-Control()

3.6 Setup de Simulação

Para todo o STE ser validado de forma a ser tirado todo o tipo de conclusões, tem que existir uma concatenação correta de todas as ferramentas de simulação.

Deste modo, para perceber essa mesma ligação entre simuladores foi criado um diagrama em que se observa os passos de todo o processo de simulação, representado na figura 3.16.

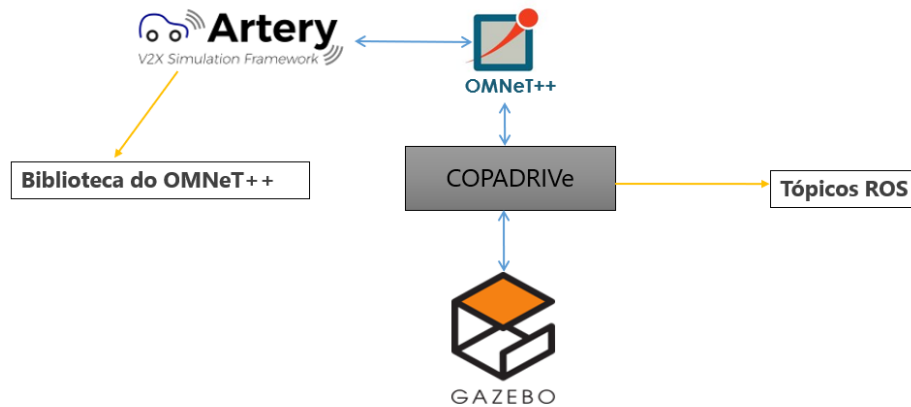


Figura 3.16: Diagrama - Setup Simulação

A partir da figura 3.16, denota-se a relação entre as várias ferramentas, com o uso do Artery para o fornecimento dos serviços relacionados com a comunicação veicular simulada pelo OMNeT++. De seguida, o OMNeT++, responsável pela comunicação entre módulos, utiliza esses mesmos serviços para enviar as mensagens que desencadeiam eventos (DENM) ou atualizam posições dos veículos (CAM) na *interface* do Gazebo, através de tópicos ROS fornecidos pelo COPADRIVe. O COPADRIVe funciona como elo de ligação entre os dois simuladores.

Capítulo 4

Sistema de Travagem de Emergência com Pelotões Cooperativos

Com o objetivo deste projeto em mente, irá ser dada uma breve visão dos pelotões cooperativos, será explicada a aplicação do STE, descrito o cenário relativo ao caso de uso do Sistema de Travagem de Emergência (STE) aplicado especificamente aos pelotões cooperativos, assim como também será demonstrado como pode ser importante a partir da semelhança com situações que ocorrem regularmente nas estradas e cidades ao redor do mundo.

4.1 Pelotões Cooperativos

Os Pelotões Cooperativos apresentam-se como uma aplicação baseada em comunicações que permitem uma partilha conjunta de informações entre diferentes veículos/RSU, de modo a que garanta a segurança necessária em qualquer projeto onde esteja inserido. Assim sendo, designa-se como uma aplicação C-ITS. Existem diversos cenários que se baseiam nos C-ITS, sendo feita uma descrição de alguns deles pela *5GAA Automotive Association* [48]. Mas mesmo tendo inúmeros cenários [14], os mais relevantes para serem estudados pertencem a dois grupos, também já apresentados anteriormente:

- Segurança rodoviária;
- Eficiência de tráfico.

Este tipo de cenários C-ITS acarretam várias vantagens, entre si a diminuição do uso do combustível, a diminuição do número de acidentes e o aumento da capacidade/eficiência rodoviária [49].

O uso do *platooning* leva ao aumento do desempenho relativo à comunicação entre veículos, existindo uma dependência relativa a essa comunicação para o bom funcionamento de todo o sistema. A distância entre veículos, o ângulo de forma a alinhar os mesmos e todo o seu estado, precisam de ser comunicadas entre todos os elementos do pelotão e para isso, a informação precisa de ser enviada no momento certo e verificada recorrentemente. Esse transformou-se no principal desafio para o uso desta tecnologia, para além dos possíveis ruídos que podem levar a perda de pacotes de mensagem, que nestes cenários pode levar a que seja fatal [50].

Na figura 4.1 está representado um dos cenários de *cooperative platooning*, verificando-se os sensores instalados nos veículos de forma a manter uma distância constante, assim como o envio da informação de redução de velocidade do veículo da frente para todos os outros veículos.

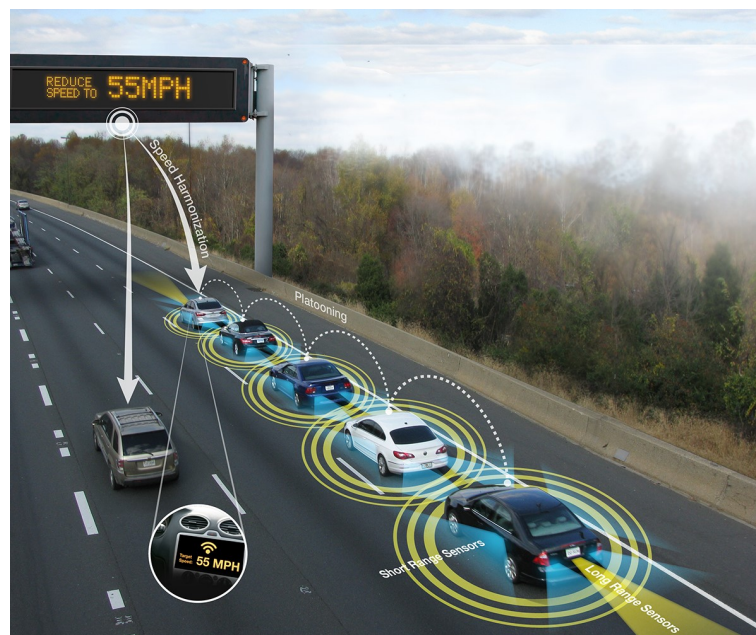


Figura 4.1: Cooperative Platooning - Exemplo [11]

A exigência destes cenários requer que sejam usados sensores e protocolos de comunicação que não permitam cometer qualquer tipo de erro. Com isto, existem alguns parâmetros que podem aumentar ainda mais as vantagens relativas ao *cooperative platooning*, mas ao mesmo tempo exigem a maior exatidão dos seus componentes, como:

- Diminuição da distância entre veículos - Proporciona um menor arrasto aerodinâmico, melhorando a eficiência do combustível;
- Aumento de veículos no pelotão - Melhor eficiência e segurança rodoviária.

Neste projeto, a aplicação do *cooperative platooning* terá dois aspectos comunicacionais, a comunicação V2V e a comunicação I2V, havendo uma representação de dois pelotões. No desenvolvimento do software referente a estas comunicações, a fiabilidade será a chave sendo que para a existência de um cenário a correr em *cooperative platooning* tem que existir uma total dependência nesse *software* para que tudo corra como o previsto e não hajam problemas de segurança. Para se visualizar este aspeto, a validação e testes de todo o *software* serão essenciais e indispensáveis, sendo apresentados os mesmos no capítulo 6.

4.2 Sistema de Travagem de Emergência (STE)

O STE no mundo real, cada vez mais se torna necessário e, com o desenvolvimento de diversas tecnologias, é muito possível de ser implementado num futuro próximo. As OBU presentes nos veículos são o primeiro passo para a realização da comunicação entre veículos e outras infraestruturas proporcionando uma larga variedade de implementações possíveis, como por exemplo o STE.

Para a sua implementação, a OBU deverá ser capaz de comunicar com as OBU's de todos os veículos na sua vizinhança assim como também com as RSU's de forma a criar uma rede de comunicação constante. Com isto, na figura 4.2 está descrita a arquitetura do STE.

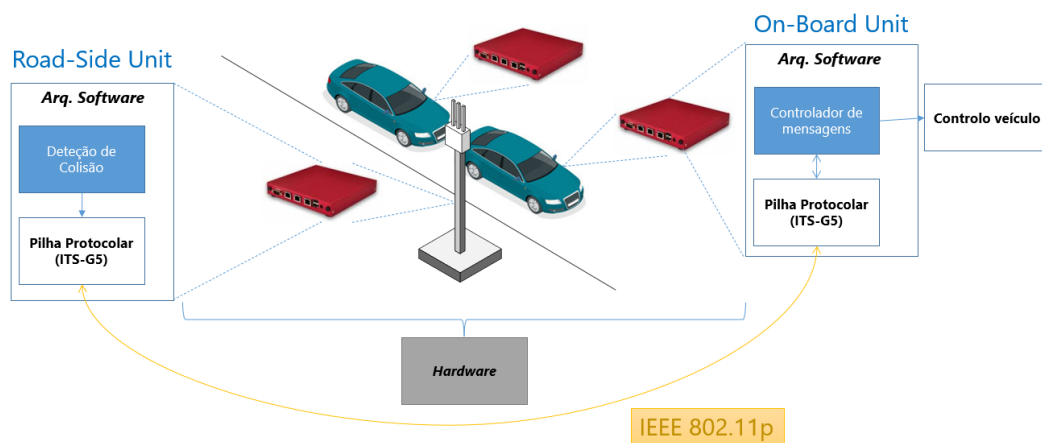


Figura 4.2: Componentes do STE e Comunicação entre dispositivos

Como observado na figura 4.2, o *hardware* do sistema é composto pela RSU, pelas OBU's e pelos veículos. Presente nas OBU's e RSU's encontra-se a pilha protocolar ITS-G5 que irá permitir a comunicação entre dispositivos através do IEEE 802.11p.

A RSU terá a capacidade de, a partir do *software* criado na própria, detetar a colisão iminente entre veículos. Essa possível colisão irá posteriormente ser comunicada a partir do IEEE 802.11p ao dispositivo presente no veículo líder, de forma a que ele consiga comunicar ao seu próprio sistema que a travagem necessita de ser feita. Para além disso, o veículo líder irá também comunicar ao outro veículo do pelotão essa informação, através do mesmo protocolo, havendo um tratamento de dados de forma semelhante ao veículo líder, resultando numa travagem de emergência. O processo de comunicação entre dispositivos pode ser observado na figura 4.3.

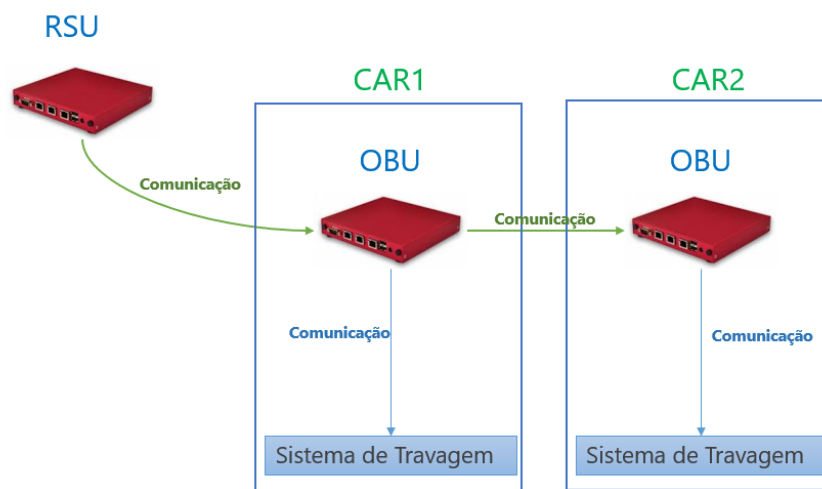


Figura 4.3: Arquitetura Geral STE

4.3 Cenário STE - Descrição

O STE, no dia-a-dia, pode ser essencial em diversas situações. Neste projeto, o contexto relativo a este caso de uso será demonstrado com a simulação da passagem de diferentes pelotões numa interseção. Na mesma, irá prevalecer uma das principais regras da condução que será a prioridade dos veículos que se apresentam pela direita. Este cenário poderá ser encontrado em todas as estradas nacionais e internacionais, resultando assim num caso universal e com possível implementação futura.

Desta forma, o objetivo principal deste cenário destaca-se pela receção de uma mensagem desencadeada por um evento, neste caso pelo risco de colisão na interseção mais próxima. Posteriormente, haverá apenas reação por parte dos veículos destinados a respeitar a prioridade imposta pela via. Na figura 4.4, está apresentado o esquema relativo ao cenário.

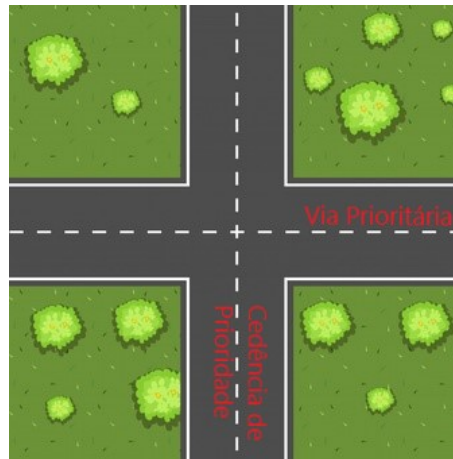


Figura 4.4: Prioridades - Interseção [12]

Para ser aplicado este cenário, o raio de ação referente a todo o evento (especificado na seção 5.3), assim como o raio de ação da própria RSU terão que envolver toda a interseção, dando tempo aos veículos de processar a informação e aplicar os processos físicos necessários para cumprir o caso de uso utilizado neste projeto.

No caso do pelotão destinado a dar prioridade à outra via, a recepção da mensagem irá despoletar uma série de ações que de acordo com o desenvolvimento feito, irá proporcionar a paragem quando se aproximar da interseção, evitando a sua colisão. Já o pelotão prioritário irá receber a informação de que poderá haver uma colisão, mas, como é prioritário, irá seguir o seu caminho sem qualquer tipo de ação realizada pelo mesmo, acreditando que a informação de paragem para os outros veículos seja entregue e os mesmos parem quando for suposto fazê-lo.

Para além das mensagens enviadas pelo RSU para desencadear o evento, também as CAM vão ser enviadas entre veículos e RSU, atualizando as informações de localização e movimento, como velocidade e ângulos relativos ao *platooning* entre veículo principal e secundário. A frequência destinada às CAM será o que vai permitir o veículo secundário parar sem colidir com o veículo principal, isto pois o intervalo de envio das CAM será pequeno, permitindo a rápida ação de um veículo relativamente ao outro.

Na tabela seguinte, a descrição sucinta deste caso de uso é realizada, desta forma, será compreendido todo o processo relativo à criação do mesmo.

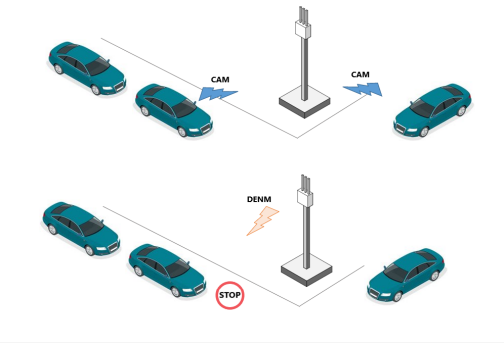
Nome do Use Case	Sistema de Travagem de Emergência.
História Use Case	Um pelotão de veículos autônomos terá que respeitar as regras de trânsito, dando prioridade e parando o pelotão de veículos ao aproximar-se de uma interseção.
Categoria	Condução Autônoma.
Ambiente	Interseção.
Descrição curta	Um pelotão de veículos autônomos recebe informação da RSU que na interseção à frente terá que travar senão irá colidir com o pelotão de veículos que também atravessará essa interseção.
Atores	Veículos dos pelotões e uma RSU.
Papel do Veículo	Um dos pelotões irá atravessar a interseção e o outro irá parar antes da interseção.
Papel da Infraestrutura	Enviar uma mensagem aos veículos de que irá haver uma possível colisão se não pararem na interseção à frente .
Objetivo	Fazer cumprir as regras de trânsito e evitar uma colisão parando um dos pelotões antes da interseção.
Requisitos	Veículos necessitam de saber a qual instante vai ser acionado o sistema de travagem.
Restrições / Presunções	Os veículos autonomamente param antes da interseção sabendo qual o instante em que devem acionar o sistema de travagem.
Âmbito Geográfico	Interseção.
Ilustração	 <p>Figura 4.5: Cenário Emergency Braking</p>
Pré-Condições	Veículos têm que estar na zona de aplicação do cenário.
Evento Principal	<ul style="list-style-type: none"> A RSU, sabendo da possibilidade de colisão irá enviar para os veículos as características da estrada, do cenário e da localização; Os veículos recebem essa informação e o pelotão que terá que dar prioridade irá parar antes de colidir com o outro pelotão.
Pós-Condições	Veículos de um pelotão irão parar e o os veículos do outro irão atravessar a interseção.
Requisitos de informação	<ul style="list-style-type: none"> Localização dos veículos e da RSU; Raio de ação da RSU para envio de mensagem; Instante em que devem enviar a mensagem para acionar o sistema de travagem.

Tabela 4.1: Descrição do Cenário

Capítulo 5

Implementação do STE

Neste capítulo será abordada a implementação do STE, no qual é prevenida uma possível colisão veicular, com essa informação a ser transmitida aos veículos de forma a que eles realizem uma travagem de emergência. A decisão de travagem será realizada autonomamente, mostrando um caso que poderá ocorrer no dia-a-dia. A estrutura de comunicação será apresentada nas próximas secções, dando também especial atenção à adaptação realizada no *DenService* de forma a que seja consumada a comunicação da RSU com os veículos. Por fim, com o objetivo de ser observado todo o processo na ferramenta de simulação robótica (Gazebo), a conexão dos processos de comunicação com o ROS será demonstrada e detalhada.

5.1 Sistema de Travagem de Emergência - Estrutura geral

O desenvolvimento deste projeto deveu-se a uma concatenação entre vários processos e vários ficheiros, desta maneira, uma explicação sobre a própria estrutura seria indispensável para uma melhor percepção de todo o trabalho realizado.

De forma a demonstrar a estrutura do caso de uso desenvolvido, foi construído um diagrama de sequência que permite de forma resumida exibir o processo.

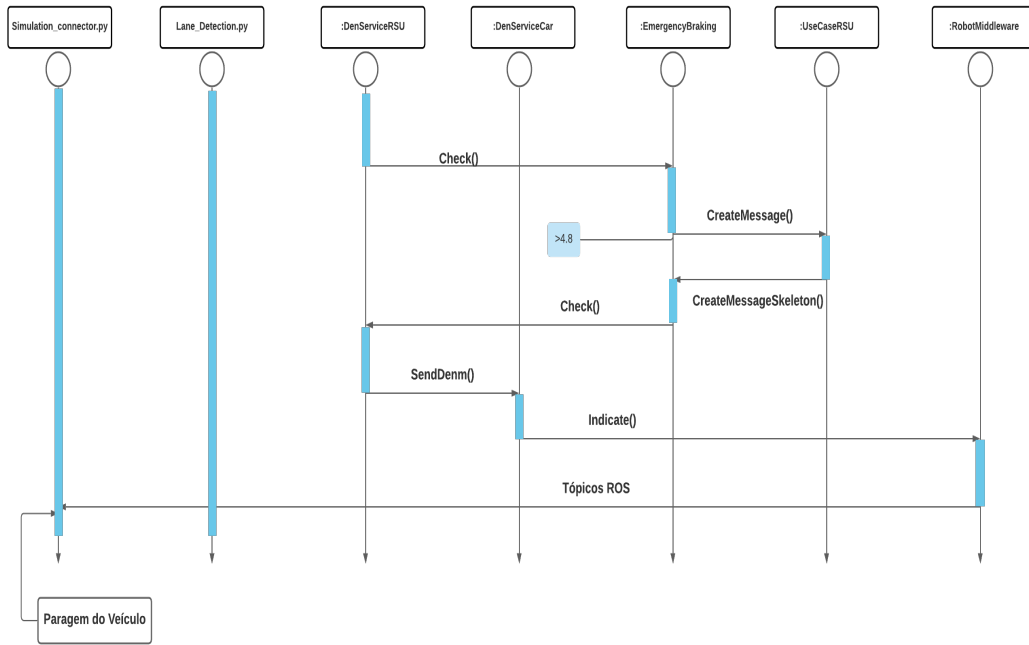


Figura 5.1: Diagrama de Sequência da implementação do STE

Como apresentado na figura 5.1, a simulação é iniciada com o auxílio de dois ficheiros *python* de forma a movimentar o veículo (*Simulation-connector.py*) e mantê-lo na trajetória predefinida (*Lane-Detection.py*).

Durante a simulação, é feito um teste constante na função *Check()* do *DenService*. Ao fazer este teste, é chamada a função *Check()* também do ficheiro *EmergencyBraking*, sendo que quando a condição do tempo for superior a 4.8s, irá ser despoletado o evento. Ao desencadear o evento, é feita uma colocação de dados na mensagem a partir dos ficheiros *UseCaseRSU* e *EmergencyBraking* (que serão explicados na secção 5.3).

A decisão de utilizar o tempo de simulação em vez da deteção de colisão de forma autónoma por parte da RSU através da localização e previsão do trajeto dos pelotões prendeu-se ao facto da existência de pouco tempo para explorar essa hipótese, tornando-se assim uma das possibilidades de implementação futura.

A passagem para o ficheiro *UseCaseRSU* é realizada na função *CreateMessage()* do *EmergencyBraking*, enquanto que o seu retorno é realizado depois de concluída a função *CreateMessageSkeleton()*. Em seguida, retorna ao *DenService* pela função *Check()* do *EmergencyBraking* para envio da mensagem DENM para o veículo. Este ficheiro é um dos principais focos referidos na secção 5.2, sendo incumbido também de verificações, como por exemplo a estação que estará a enviar a mensagem DENM. Esta é uma necessidade do projeto, pois apenas a RSU deve enviar DENM.

Depois de feita a comunicação e enviada a mensagem, é a vez do *RobotMiddleware* tratar da receção da mesma e envio, por tópicos ROS, de alguns dos dados relativos

aos veículos e à RSU, sendo que a passagem para este ficheiro é realizada pela função *Indicate()*, neste caso relativa ao *DenService* do veículo. Imediatamente a seguir de enviada a DENM por parte da RSU, verificado no tópico ROS referente à receção de mensagem, entra em ação o ficheiro *python* alusivo ao controlo do veículo, levando à paragem do mesmo.

Simultaneamente a todo este processo, o envio de CAM implementado pelo CO-PADRIve seguirá o mesmo cronograma temporal, com a passagem pelos mesmos ficheiros, apenas o seu tratamento e envio é realizado pelo *CaService*, sendo a comunicação feita também a partir de tópicos ROS, existindo o controlo do veículo pelo ficheiro *Simulation-Connector.py*. A figura 5.2 apresenta um diagrama temporal de envio de DENM e CAM entre infraestrutura e neste caso, entre o *Car1* e o *Car3* em forma de exemplo, visto que mais 2 veículos estarão presentes no cenário, enviando também CAM e recebendo também DENM.

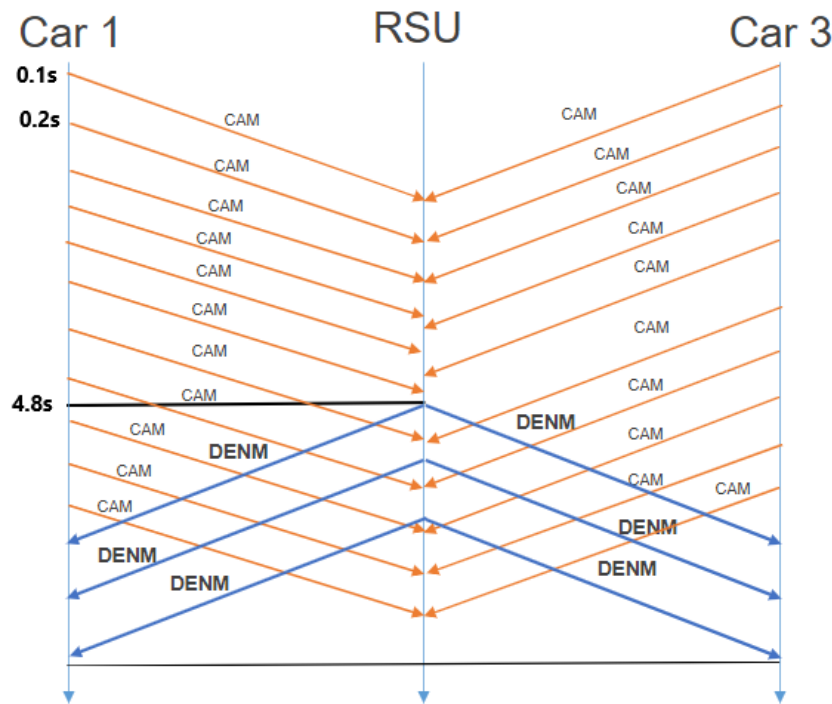


Figura 5.2: Diagrama Temporal - Envio de Mensagens CAM e DENM

5.2 Adaptação do ficheiro DenService

O *DenService* é um processo intermédio que irá processar o envio da mensagem, assim como processos iniciais da receção da mesma a partir da função *Indicate()*, responsável pela receção e envio de sinais quando uma mensagem é recebida. Este envio de sinais irá permitir a ligação ao *RobotMiddleware* e a confirmação de que a

DENM foi recebida. Para alcançar o objetivo imposto para este trabalho, foi necessário alterar o mesmo relativamente ao que era fornecido pelo Artery, mantendo-se as mesmas bases de funcionamento.

Inicialmente, houve uma necessidade de alterar o *timer* imposto pelo serviço, de maneira a que seja utilizado o *timer* que permite uma sincronização entre o OMNeT++ e o Gazebo pois, como já foi abordado, os dois têm *timers* a funcionar de maneira diferente. Deste modo, utilizou-se o *timer* desenvolvido pelo COPADRIVe, estabelecido no *Facilities* que já permitia uma integração e funcionamento em simultâneo dos dois simuladores. A chamada desse *timer* pode ser vista na figura 5.3.

```
void DenService::initialize()
{
    ItsGSBaseService::initialize();
    mTimer = &getFacilities().get_const<Timerosomnet>();
    mMemory.reset(new artery::den::Memory(*mTimer));

    subscribe(storyboardSignal);
    subscribe(denmReceivedSignal);
    initUseCases();
}
```

Figura 5.3: DenService - Chamada do Timer

Seguidamente, foi também introduzida uma função de verificação de estação que iria enviar a DENM, necessária para apenas ser a RSU a fazê-lo como podemos ver na figura 5.4.

```
void DenService::verifyStation()
{
    if(this->getParentModule()->getId() == 17)
    {
        sendFromRSU=true;
        rsuId=this->getParentModule()->getId();
    }
    else
    {
        sendFromRSU=false;
    }
}
```

Figura 5.4: DenService - VerifyStation()

Esta função torna-se uma das funções mais importantes do código, sendo chamada e abordada na maior parte do mesmo, havendo uma dependência de várias funções relativas a ela. Como é possível ver na função mais importante do *DenService* que trata do envio da DENM como está demonstrado na figura 5.5.


```

void DenService::sendDenm(vanetza::asn1::Denm&& message, vanetza::btp::DataRequestB& request)
{
    verifyStation();

    if(sendFromRSU == true){
        DenmObject obj { std::move(message) };
        emit(denmSentSignal, &obj);
        auto size = obj.asn1().size();
        double send_time = simTime().dbl();
        std::cout << "DenService::sendDenmRSU (2): " << send_time <<std::endl;

        using namespace vanetza;
        using DenmConvertible = convertible::byte_buffer_impl<asn1::Denm>;
        std::unique_ptr<geonet::DownPacket> payload { new geonet::DownPacket };
        std::unique_ptr<vanetza::convertible::byte_buffer> denm { new DenmConvertible ( obj.shared_ptr() ) };
        payload->layer(0silayer::Application) = std::move(denm);
        this->request(request, std::move(payload));
    }
}

```

Se for a RSU envia a DENM

Figura 5.5: DenService - SendDenm()

Relativamente ao *trigger* evidenciado pelo serviço, o mesmo é utilizado sem sofrer quaisquer alterações (de forma a que possa ser utilizado posteriormente para mais casos de uso que se possam implementar), mas a sua chamada fará o mesmo esperar que seja desencadeado o evento estabelecido pelo ficheiro do *EmergencyBraking*. Essa parte será também referida com especial atenção no capítulo 7.

```

void DenService::trigger()
{
    mMemory->drop();

    for (auto& use_case : mUseCases) {
        use_case.check();
    }
}

```

Figura 5.6: DenService - Trigger()

Finalmente, a função de *request* já fornecida torna-se inutilizável de forma a usar todas as especificações e pedidos de acordo com o estabelecido no *EmergencyBraking* e no *UseCaseRSU*, essas especificações que irão ser abordadas na secção 5.3.

5.3 STE - Especificação e Funcionalidades

Em seguida, para este caso de uso funcionar e todas as informações corretas serem transmitidas, uma especificação completa e organizada necessita de ser detalhada. Para o fazer, foram utilizados 2 ficheiros diferentes, de seu nome *UseCaseRSU* e *EmergencyBraking*.

No primeiro, é detalhado de forma mais geral (de forma a ser usado noutros casos de uso) a informação relativa ao envio das DENM e à identificação da estação que irá enviar essa mesma informação, neste caso, a RSU.

```

vanetza::asn1::Denm UseCaseRSU::createMessageSkeleton()
{
    vanetza::asn1::Denm message;
    message->header.protocolVersion = 1;
    message->header.messageID = ItsPduHeader__messageID_denm;
    message->header.stationID = mService->getParentModule()->getId();

    auto action_id = mService->requestActionID();
    message->denm.management.actionID.originatingStationID = action_id.originatingStationID;
    message->denm.management.actionID.sequenceNumber = action_id.sequenceNumber;

    int ret = 0;
    const auto taiTime = 0;

    ret += asn_long2INTEGER(&message->denm.management.detectionTime, taiTime);
    ret += asn_long2INTEGER(&message->denm.management.referenceTime, taiTime);
    assert(ret == 0);
    message->denm.management.eventPosition.altitude.altitudeValue = AltitudeValue_unavailable;
    message->denm.management.eventPosition.altitude.altitudeConfidence = AltitudeConfidence_unavailable;

    message->denm.management.eventPosition.longitude = 60;

    message->denm.management.eventPosition.latitude = 100;

    message->denm.management.eventPosition.positionConfidenceEllipse.semiMajorOrientation = HeadingValue_unavailable;
    message->denm.management.eventPosition.positionConfidenceEllipse.semiMajorConfidence = SemiAxisLength_unavailable;
    message->denm.management.eventPosition.positionConfidenceEllipse.semiMinorConfidence = SemiAxisLength_unavailable;

    message->denm.location = vanetza::asn1::allocate<LocationContainer_t>();
    message->denm.location->eventSpeed = vanetza::asn1::allocate<Speed>();
    message->denm.location->eventSpeed->speedValue = 0;

    message->denm.location->eventSpeed->speedConfidence = SpeedConfidence_equalOrWithinOneCentimeterPerSec * 3;
    message->denm.location->eventPositionHeading = vanetza::asn1::allocate<Heading>();
    message->denm.location->eventPositionHeading->headingValue = 0;

    message->denm.location->eventPositionHeading->headingConfidence = HeadingConfidence_equalOrWithinOneDegree;

    auto path_history = vanetza::asn1::allocate<PathHistory_t>();
    asn_sequence_add(&message->denm.location->traces, path_history);

    return message;
}

```

Colocar ID da estação que envia mensagem na mensagem DENM

Latitude e Longitude do evento

Especificação da localização do evento

Figura 5.7: UseCaseRSU - CreateMessageSkeleton()

Como visualizado na figura 5.7, é colocado na mensagem enviada um conjunto de parâmetros que permite caracterizar alguns dos *containers* obrigatórios para envio de DENM, como o *management container* e o *location container*. Como dito anteriormente, estes são alguns parâmetros universais, ou seja, possíveis para usar com outros casos de uso. A possibilidade de escolha destes parâmetros estão explícitos no BSP das DENM [6].

Relativamente ao segundo, é especificado o cenário utilizado neste projeto, com a caracterização do evento em si, esta especificação está também de acordo com o BSP das DENM fornecido pelo ETSI [6].

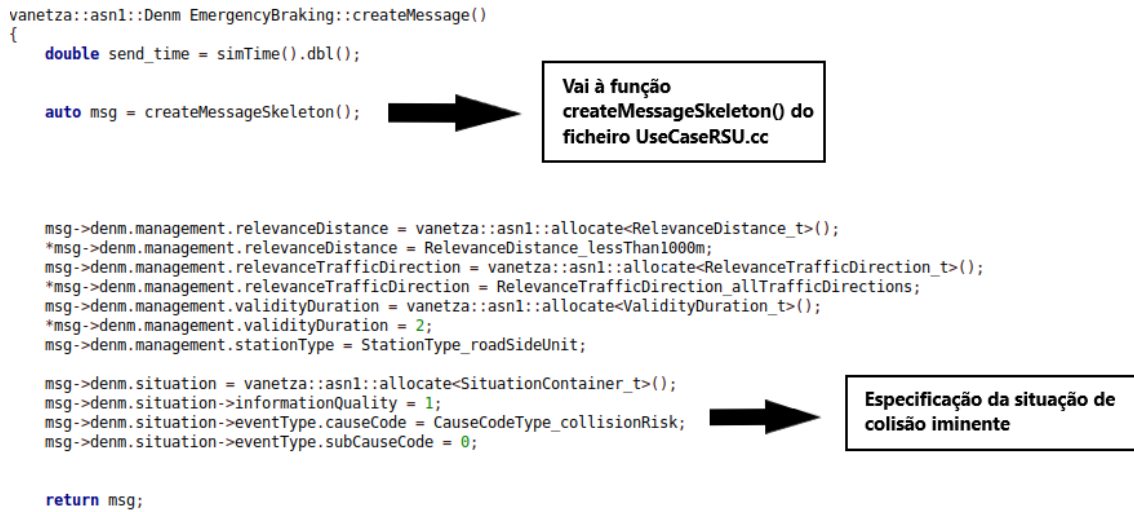


Figura 5.8: EmergencyBraking- CreateMessage()

Como observado na figura 5.8, é colocada informação na mensagem enviada, desta vez de forma mais completa. A informação dada aos *containers* é mais específica para o caso de uso utilizado, com a colocação da duração do evento, de qual a causa, assim como a distância ao mesmo. Para além disso, é dito qual é o tipo de estação que envia a mensagem e o código que especifica ainda mais o caso de uso que, como verificado, se reflete em um caso de *Collision Risk*.

Outro dos pontos muito importantes relativos ao *EmergencyBraking*: a comunicação entre estações tem que ser especificada de igual forma. Para isso, foi construída uma função que fornece todas as informações relativas à comunicação. Essa função, de seu nome *CreateRequest()* pode ser observada na figura 5.9.

```

vanetza::btp::DataRequestB EmergencyBraking::createRequest()
{
    namespace geonet = vanetza::geonet;
    using vanetza::units::si::seconds;
    using vanetza::units::si::meter;

    using namespace vanetza;

    vanetza::btp::DataRequestB request;
    request.gn.traffic_class.tc_id(1);
    request.destination_port = btp::ports::DENM;
    request.gn.its_aid = aid::DEN;
    request.gn.transport_type = geonet::TransportType::GBC;
    request.gn.communication_profile = geonet::CommunicationProfile::ITS_G5;

    geonet::Area destination;
    geonet::Circle destination_shape;
    destination_shape.r = 1000.0 * meter;
    destination_shape = destination_shape;

    destination.position.latitude = (120) * vanetza::units::degree;
    destination.position.longitude = (45.8) * vanetza::units::degree;
    request.gn.destination = destination;

    return request;
}

```

Figura 5.9: EmergencyBraking - CreateRequest()

Desta forma, como constatado pela figura 5.9, esta função estrutura todo o tipo de ligações para existir uma comunicação entre as estações, assim como também é definida a área de destino, a sua forma e a latitude e longitude posicional da mesma. Depois de concluídos estes passos, a mensagem tem todos os parâmetros necessários para ser enviada, ficando completa a implementação do caso de uso.

5.4 Ligação ROS/Gazebo - Paragem dos veículos

A via com necessidade de cedência de prioridade reúne 2 veículos totalmente autónomos que se deslocam a uma velocidade de cerca de 50 km/h, especificada num dos ficheiros que irão ser apresentados de seguida.

Antes disso, o método de deslocamento dos veículos abordado baseia-se num sistema de deteção de linha, utilizado nos dois pelotões. Esse sistema está assente na cor utilizada pela linha, detetando a mesma através de processos relativos a *OpenCV*. Devido à existência de duas linhas que se irão cruzar na interseção, a cor não poderá ser a mesma, senão o pelotão não se manterá na rota. Com isto, a figura 5.10 mostra o processo de escolha da cor para os diferentes pelotões.

```
def callback(self,data):
    global lines_old
    def select_rgb_white_yellow(image):
        # white color mask
        lower = np.uint8([250, 250, 250])
        upper = np.uint8([255, 255, 255])
        white_mask = cv2.inRange(image, lower, upper)
        # yellow color mask
        lower = np.uint8([170, 190, 0])
        upper = np.uint8([255, 255, 255])
        yellow_mask = cv2.inRange(image, lower, upper)
        # combine the mask
        mask = cv2.bitwise_or(white_mask, yellow_mask)
        masked = cv2.bitwise_and(image, image, mask = mask)
        return masked
    def convert_hsv(image):
        return cv2.cvtColor(image, cv2.COLOR_RGB2HSV)
    def convert_hls(image):
        return cv2.cvtColor(image, cv2.COLOR_RGB2HLS)
    def select_white_yellow(image):
        converted = convert_hsv(image)
        # white color mask
        #Lower = np.uint8([ 70, 20, 85])
        #upper = np.uint8([80, 30, 100])
        #white_mask = cv2.inRange(converted, Lower, upper)
        # yellow color mask

        lower = np.uint8([0, 120, 0])
        upper = np.uint8([170, 180, 150])

        #Lower = np.uint8([0, 120, 0])
        #upper = np.uint8([170, 255, 130])
        yellow_mask = cv2.inRange(converted, lower, upper)
        # combine the mask
        mask = yellow_mask # cv2.bitwise_or(white_mask, yellow_mask)
        return cv2.bitwise_and(image, image, mask = mask)
```

Figura 5.10: LaneDetection - callback()

A ação desencadeada pelo evento de risco de colisão será executada depois de recebida a informação por parte da RSU, através de tópicos ROS. Para isto, um ficheiro de nome *Simulation-connector.py* é utilizado. Neste ficheiro, é realizada a subscrição ao tópico ROS *"car1/RXNetwork-OMNET"*, sendo que quando for colocada uma mensagem nesse mesmo tópico, irá ser chamada uma função de seu nome *EmergencyBraking*.

Com isto, de forma a que o veículo pare e seja respeitada a publicação da informação no tópico ROS, a função denominada por *EmergencyBraking* é chamada, verificando se alguma das mensagens enviadas para esse tópico são delegadas pela RSU, se esse for o caso, o veículo trava até se imobilizar. Na figura 5.11 encontra-se representada essa função.

```
def EmergencyBraking(data):  
  
    global EmergencySTOP  
    global car_name_TV  
    print 'Emergency Braking!!!'  
    print (data.car_name)  
    if (data.car_name == "/rsu/"):   
        EmergencySTOP = ON  
        print 'Emergency Braking IF!!!'
```

Figura 5.11: Simulation-Connector.py - EmergencyBraking()

5.5 Conclusões e pontos a salientar

Quanto à implementação do STE, uma precisa orquestração dos ficheiros revelou-se necessária com a colocação de toda a informação do evento na mensagem enviada. Com isso, foi realizada uma adaptação ao serviço de envio de DENM assim como criado um ficheiro (*EmergencyBraking*) que permitiu uma especialização posterior dessa mesma informação. A criação deste ficheiro e a sua dependência pelo *DenService*, abre as portas a possíveis implementações futuras de outros casos de uso dependentes também desse mesmo *DenService*.

Capítulo 6

Validação e Resultados Finais

Nesta secção, todos os resultados serão analisados e detalhados, em especial resultados referentes ao caso de uso implementado. Posteriormente, será realizada uma validação e avaliação do STE, usando diferentes parâmetros.

6.1 Descrição da Experiência

Para os serviços utilizados neste projeto, como o *DenService* e o *CaService*, será então usada a frequência de 5.9GHz e o *channel* 180, que como podemos ver na figura 6.1 está associada aos serviços, assim como também é a frequência de operação do IEEE 802.11p.

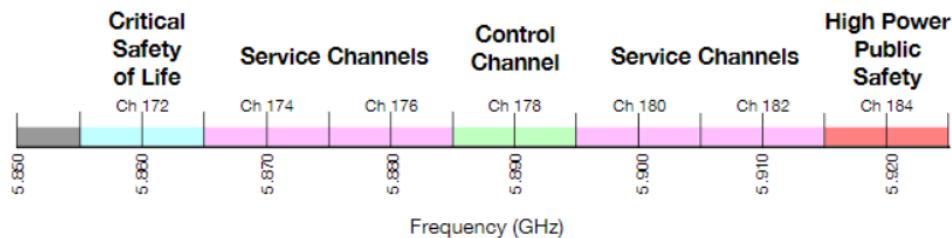


Figura 6.1: Espectro - DSRC [13]

Tal como já foi abordado anteriormente, o primeiro passo para a visualização do projeto final será o desencadeamento do evento de acordo com o caso de uso desenvolvido e, para isso, a utilização do tempo após o início da simulação torna-se

essencial, como se visualiza na figura 6.2.

```
void EmergencyBraking::check()
{
    double send_time = simTime().dbl();
    auto action_id = mService->requestActionID();

    if(send_time > 4.8){
        if(action_id.originatingStationID == 17){
            auto message = createMessage();
            auto request = createRequest();
            mService->sendDenm(std::move(message), request);
        }
    }
}
```

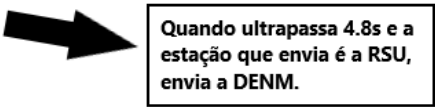


Figura 6.2: EmergencyBraking - Check()

Neste caso, a variável *send-time* reflete o tempo do OMNeT++, em que a mensagem só será enviada quando o tempo for superior a 4.8s. Outra das condições referentes a esse envio e retratadas na figura é a estação de origem, ou seja, a estação que enviará a mensagem terá que ter o *ID* 17, a única estação que é suposto enviar DENM, a RSU.

Da mesma forma que as mensagens estão a ser enviadas no OMNeT++, o cenário no Gazebo prossegue em simultâneo, com o envio das mensagens CAM entre veículos e também RSU a atualizar todo o estado do sistema e as suas posições. Como cenário principal, os veículos entram em aceleração no início da simulação, deslocando-se a cerca de 50 km/h ao chegar à interseção sendo que, mais à frente, foram validadas outras velocidades para perceção de condições relativas à comunicação e posterior travagem.

Relativamente aos parâmetros iniciais, o envio de mensagens CAM tem uma frequência de 100 Hz enquanto que as mensagens DENM têm uma frequência de repetição de 33 Hz depois de desencadeado o evento.

De forma a perceber melhor os parâmetros do cenário base, apresenta-se uma tabela com todos os pontos iniciais referentes a esse mesmo cenário:

Velocidade-Alvo (km/h)	Inst. Envio de Mensagem (s) (T.OMNeT)	Frequência - Envio de CAM (Hz)	Frequência - Envio de DENM (Hz)	Temp. Simulação (s) (T.Real)
50	4.8	100	33	≈ 120

Tabela 6.1: Parâmetros - Cenário Base

Com a paragem a ser feita pelos veículos depois de recebida a mensagem enviada pela RSU, na interface gráfica do OMNeT++ esse envio pode ser visto na figura 6.3.

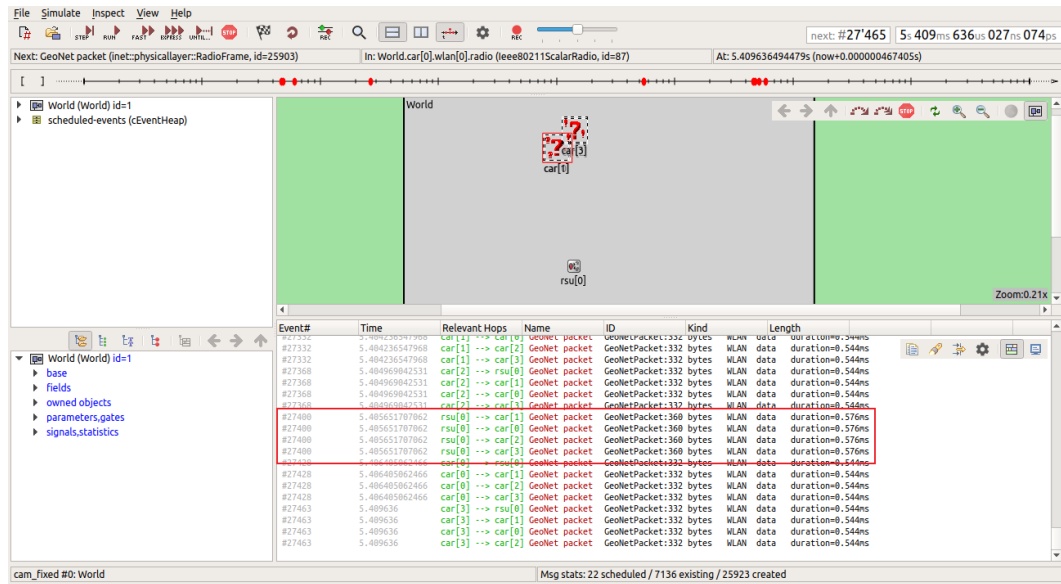
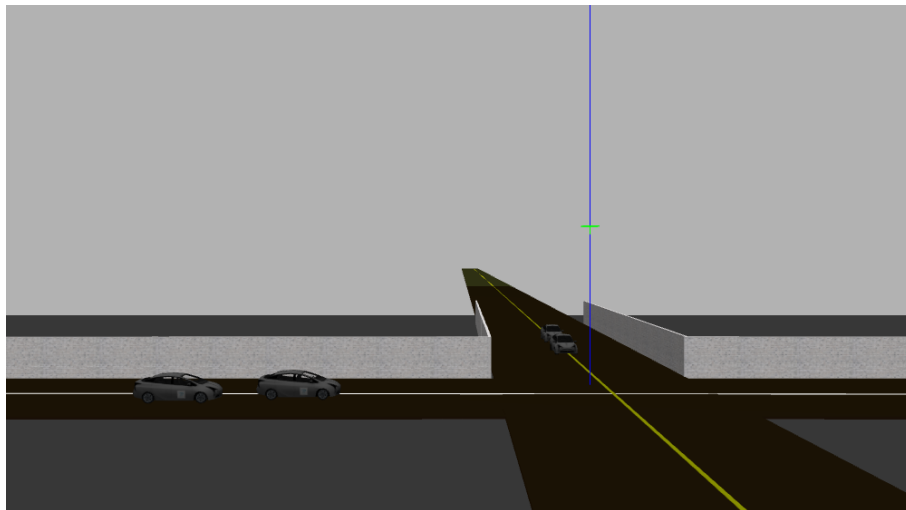


Figura 6.3: Envio das mensagens - OMNeT++

Enquanto isso, o veículo *Car1* trava. Pouco depois, o *Car2* começa também a travar, parando antes de existir qualquer colisão, como observado na figura 6.4.



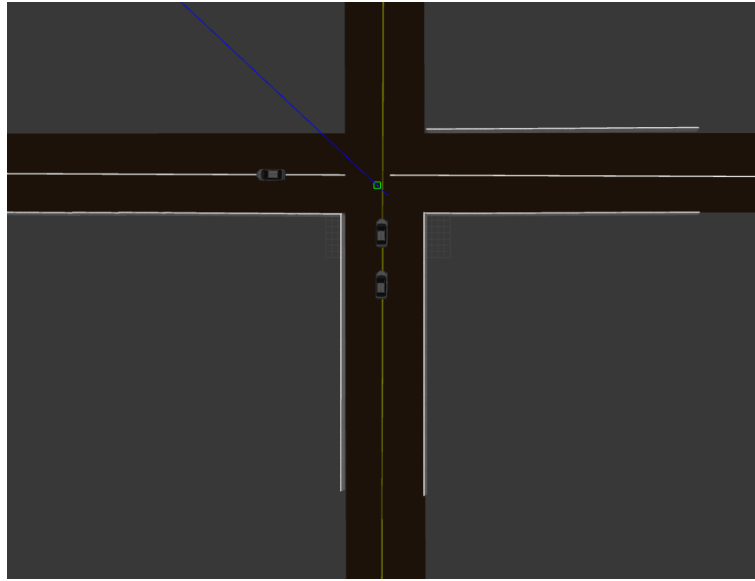


Figura 6.4: Emergency Braking - Cenário

6.2 Resultados e Validação do Cenário-Principal

Foi então realizado um teste relativo à paragem dos veículos quando os mesmos aceleram até aos 50 km/h e depois recebem a mensagem DENM para iniciar a travagem. As figuras 6.5 e 6.6 mostram a velocidade de todos os veículos presentes no cenário.

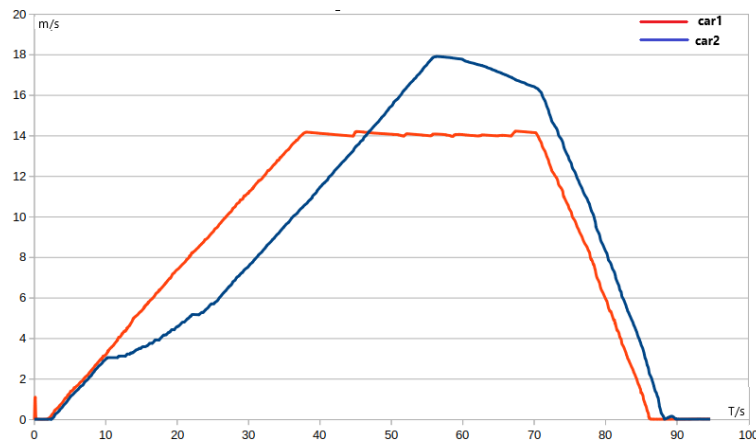


Figura 6.5: Velocidades do Car1 e Car2 a 50 km/h

Observando a figura 6.5, percebe-se que o *Car2* referente ao *follower* demora mais que o veículo líder do pelotão a efetuar a aceleração para os pedidos 50 km/h (14 m/s), sendo que para compensar e tentar acompanhar posteriormente o líder, a sua velocidade ao fim de cerca de 48 segundos é superior à do mesmo. Seguidamente, quando é enviada a mensagem de paragem de emergência, aos 71 segundos, existe uma descida proporcionalmente igual entre os dois veículos em que, devido à maior

velocidade do *car2* no momento, o mesmo demora mais tempo a parar, não havendo colisão entre o pelotão dado que existe uma distância de segurança que permite que isso aconteça.

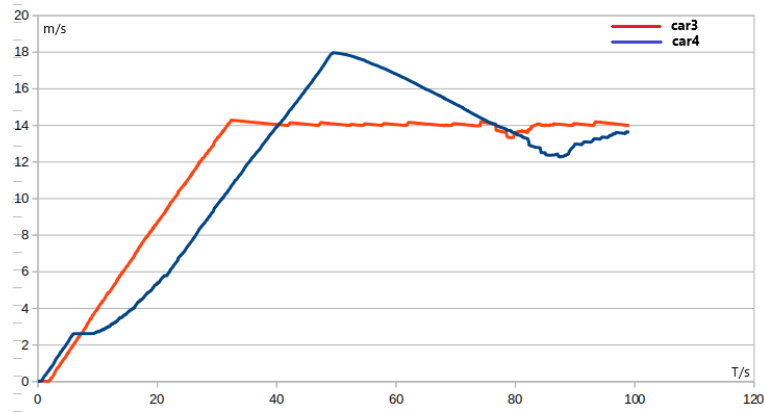


Figura 6.6: Velocidades do Car3 e Car4 a 50 km/h

Já relativamente ao outro pelotão representado pelo *Car3* e *Car4*, como não existe nenhum envio de mensagem para o mesmo parar (pois existe uma prioridade na via onde se desloca esse pelotão) o veículo *Car3* entra em aceleração tal como o *Car1* e o veículo *Car4* acompanha o mesmo tal como o *Car2* acompanha o *Car1*, tentando manter uma velocidade fixa nos 50 km/h.

6.3 Resultados para diferentes velocidades

Começando pelos processos relacionados com a diferença de velocidade, foi mantida uma distância de cerca de 100 metros para a interseção assim como uma frequência de envio de cerca de 10 ms. Com isto, o objetivo foi verificar a diferença que iria existir na travagem do veículo, na distância para o carro líder do pelotão após travagem assim como no tempo de envio da mensagem em relação ao início da simulação.

Velocidade-Alvo (km/h)	Inst. Envio de Mensagem (s)	T. Travagem (s)	Dist. <i>Car2</i> - <i>Car1</i> Após Travagem (m)
50	4.8	17.7	6
60	4.5	23.78	18.3
70	3.8	25	25.2

Tabela 6.2: Parâmetros - Cenário para diferentes velocidades

Como observado na tabela 6.2, o aumento da velocidade aumenta o tempo de travagem do veículo. Com isto entende-se que com o aumento de velocidade a janela de

envio de mensagem diminui consideravelmente, tendo assim que existir uma grande fiabilidade de envio de mensagem.

Outro dos pontos analisados prende-se na distância do veículo não líder do pelotão para o veículo líder do pelotão, com uma análise mais detalhada na figura 6.7.

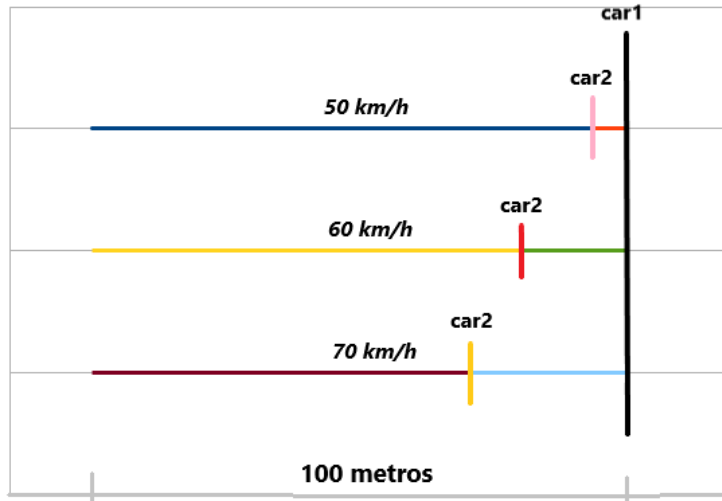


Figura 6.7: Análise da distância entre o Car2 e Car1 após travagem

Como contemplado na tabela 6.2 e na figura 6.7, a distância entre veículos do pelotão sofre uma variação bastante significativa, pois a velocidade é maior e o tempo de aproximação do *Car2* ao *Car1* é mais longo, não existindo uma aproximação atempada quando é dada a ordem de paragem.

Em conclusão, estes resultados são influenciados pelo padrão de aceleração e do controlador do *platooning*. Mostra essencialmente que poderemos observar diferentes comportamentos de paragem pelo pelotão consoante a variação de parâmetros de cenário (neste caso, a velocidade-alvo de cruzeiro). Não servindo para avaliar diretamente a performance do STE, esta análise é informativa em relação à variedade de comportamento que podemos observar.

De forma a explicar melhor esta observação, foram realizados gráficos de velocidade com o tempo para os valores impostos de 60 km/h e 70 km/h, como pode ser visto na figura 6.8 e na figura 6.9, respetivamente.

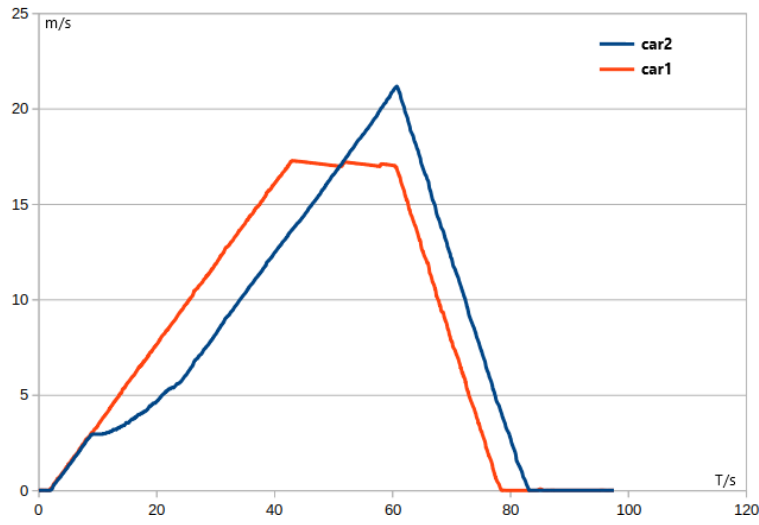


Figura 6.8: Velocidades do Car1 e Car2 a 60 km/h

Com isto, é visualmente possível verificar que no caso da velocidade requerida ser 60 km/h, a travagem é realizada quando o *Car2* se encontra em processo de aproximação do *Car1* de forma a manter o pelotão, por isso, quando é dada a ordem de paragem (60s), o *Car2* se encontra mais afastado relativamente ao *Car1* do que quando a velocidade requerida é de 50 km/h, no qual a aproximação já foi realizada.

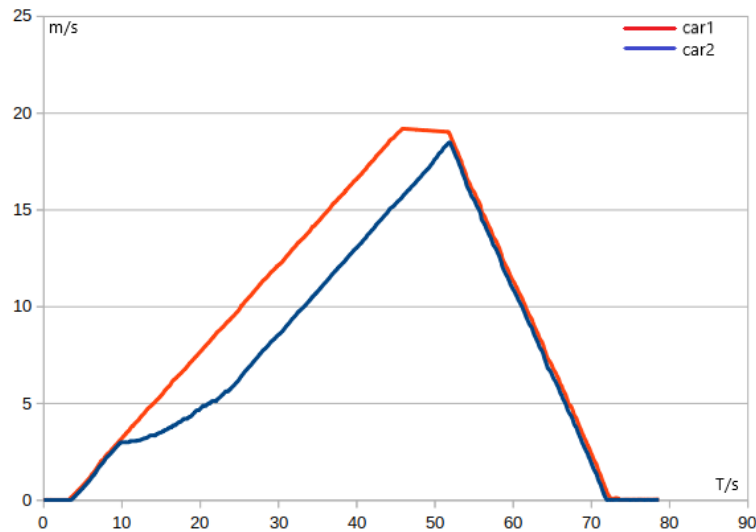


Figura 6.9: Velocidades do Car1 e Car2 a 70 km/h

Em relação a quando a velocidade imposta é de 70 km/h, o processo de aceleração ainda está a ocorrer no *Car2* por isso, ainda não teve tempo nem velocidade suficiente para chegar a uma distância mais próxima do *Car1*, quando é dada a ordem de paragem depois de enviada a DENM.

6.4 Resultados para diferentes frequências de envio de mensagens

Um dos principais testes para avaliação e validação do cenário do STE devido à capacidade de percepção do sucesso ou insucesso da comunicação no STE, foi o teste de variação da frequência de envio de mensagens, desta forma, mantendo os 50 km/h e o tempo em que a mensagem é enviada, varia-se apenas a frequência de envio de mensagem. Avaliando então essas possibilidades, podemos concluir que vão existir comportamentos diferentes e atraso de informação para valores diferentes de frequência, apresentando de seguida alguns desses comportamentos.

Frequência(Hz)	T. Travagem(s)	Dist. <i>Car2</i> - <i>Car1</i> Após Travagem (m)
100	17.7	≈ 6
1.25	20.42	≈ 4
1	20.7	≈ 0

Tabela 6.3: Parâmetros - Cenários para diferentes frequências

- Para uma frequência de 100Hz como descrito na tabela, o funcionamento do processo é executado perfeitamente, mantendo uma distância de segurança e parando de forma suave.
- Para uma frequência de 1.25Hz o panorama já começa a ser diferente, com atrasos no envio das mensagens, o *Car2* sente uma maior dificuldade em acompanhar o carro líder, havendo algumas mudanças na trajetória. Para além disso, existe uma maior demora na travagem dos veículos devido ao atraso na receção da mensagem DENM, levando a que o *Car2* se aproxime mais do carro líder ao fim da travagem, ficando a poucos metros de haver uma colisão entre o pelotão. Mas, com esta frequência ainda o processo é executado satisfatoriamente.
- Por fim, para uma frequência de 1Hz, mantêm-se os atrasos nos envios da mensagem, desta vez com um atraso maior. O *Car2* tem também dificuldade em acompanhar o líder, assim como também a travagem demora mais por parte dos veículos. Como o atraso em relação à comunicação entre o carro líder e o seu seguidor é tão grande, quando o *Car1* trava e pára, o seu seguidor (*Car2*) acaba por colidir com o mesmo, parando posteriormente.

6.5 Conclusões e pontos a salientar

De modo a testar o cenário e a implementação feita pelo STE, foi feita uma série de testes utilizando diferentes velocidades, assim como diferentes frequências de

envio de mensagens. Estes testes são principalmente relativos ao comportamento do pelotão de veículos antes, durante e após a travagem de emergência imposta para evitar a colisão.

Com isto, os resultados são satisfatórios, com o funcionamento correto de todo o cenário, sendo atingido o objetivo proposto inicialmente.

Alguns dos pontos a salientar prende-se no facto de todo o processo ser realizado tendo em mente os parâmetros relativos à travagem e aos efeitos da mesma no pelotão, como por exemplo a distância entre veículos (após travagem) e o tempo de travagem.

A distância entre o *Car2* e o *Car1* encontra-se um pouco inviabilizada devido a não conseguir haver uma aproximação atempada entre veículos do pelotão, comprometendo os resultados referentes à distância. Este acontecimento é devido ao controlador de *platooning* necessitar de uma aceleração inicial e o desenho do cenário não ter sido corretamente realizado para este aspeto.

Uma das soluções possíveis seria aumentar a distância inicial relativamente à interseção, de forma a que quando chegasse à mesma os dois veículos estivessem à mesma velocidade.

Relativamente a um dos principais pontos, a frequência de envio de mensagens, a alta dependência do sistema relativamente à mesma foi revelada, necessitando de uma alta frequência para não existirem problemas na atualização de todo o cenário. A baixa frequência revelou problemas que levaram à colisão entre veículos e ao atraso de informações, invalidando o STE.

Capítulo 7

Conclusões

Relativamente ao STE, as comunicações e as suas frequências revelaram-se como as peças mais importantes de toda a simulação, criando uma variação muito mais drástica que todos os outros parâmetros. A dependência das comunicações na manutenção da cooperação entre veículos é uma das razões para não existir uma aplicação destes conceitos no dia-a-dia e é o principal obstáculo a ser ultrapassado.

Os Pelotões Cooperativos poderão ser bastante desenvolvidos e explorados num futuro próximo, mas neste momento ainda se prende com algumas teorias envolvidas com a segurança e com a política que fazem com que não haja uma maior aplicação do mesmo. Neste aspeto, a indústria estará à espera de novos desenvolvimentos relativos à tecnologia nas áreas de comunicação.

No caso dos múltiplos cenários possíveis de implementar, esta grande variedade permite num futuro existir uma maior exploração neste campo, usando os ITS como ponto de partida e baseando-se então na comunicação V2X fornecida por uma das normas mais bem aceites no panorama europeu, o ETSI ITS-G5.

Relativamente ao projeto em si, foi desenhado e implementado um Sistema de Travagem de Emergência aplicado a um cenário de interseção simulado em plataformas de simulação. Muitas dificuldades foram encontradas, desde a instalação de todos os softwares até ao material utilizado para os instalar, com muitos pormenores a tornarem-se verdadeiros obstáculos diminuindo o ritmo de trabalho. Mas, mesmo com todos estes problemas de processamento e desenvolvimento, o objetivo proposto foi devidamente alcançado, o que levou à possibilidade de existirem diversos testes e validações ao próprio cenário.

Em relação a essas mesmas validações, existiu uma flexibilidade relativamente às diferentes velocidades impostas. Quanto à compreensão das limitações, a baixa frequência de envio de mensagens, reflete-se como a maior dificuldade a ultrapassar como já foi referido, sendo que uma variação por mais mínima que seja pode comprometer todo o sistema. Neste caso, com 100 Hz não existiu colisão mas a partir de uma frequência de envio de mensagens mais baixa como 1 Hz já existiu colisão, não conseguindo o STE evitar a mesma.

Por outro lado, agora quanto às tecnologias utilizadas, o Gazebo correspondeu bastante bem ao pedido por este projeto, com um desenho de uma interseção para aplicação do STE bem conseguido. O mesmo se pode dizer do OMNeT++, que para a comunicação entre veículos e infraestrutura se revelou como totalmente satisfatório, havendo uma boa conexão com o Gazebo, formando uma *framework* verdadeiramente bem estruturada.

7.1 Trabalho Futuro

O grande desenvolvimento e a sua grande flexibilidade irá permitir a implementação de vários cenários diferentes, explorando as comunicações existentes, assim como também os seus serviços. Essencialmente, este projeto, irá abrir novos horizontes para várias implementações futuras e novos projetos serão criados a partir do mesmo. Relativamente ao Sistema de Travagem de Emergência (STE), o seu futuro poderá passar pela sua implementação de forma física a partir das OBU's instaladas nos veículos a comunicar com uma infraestrutura apenas simulada. Com isto, é utilizado assim um cenário mais palpável que permita tirar resultados mais concretos do que poderá ser usado no futuro. Relativamente ao mesmo, também existirão problemas que apenas fisicamente são observados, sendo essa a principal razão para que nem todos os cenários simulados sejam implementados.

Referências

- [1] “Integrity and authentications for service security in vehicular ad hoc networks (vanets): A review - scientific figure on researchgate.” https://www.researchgate.net/figure/Vehicular-ad-hoc-network-VANETS_fig1_352787389. [Citado nas páginas vii e 6]
- [2] B. Vieira, “A simulation approach for increased safety in advanced c-its scenarios,” Master’s thesis, Porto, Portugal, 2019. [Citado nas páginas vii, 7, 15, 17, 18 e 20]
- [3] “Comparing limeric and dcc approaches for vanet channel congestion control.” https://www.researchgate.net/figure/Comparison-of-the-protocol-stacks-in-the-US-called-WAVE-and-the-EU-stack-developed_fig1_286562334. [Citado nas páginas vii e 8]
- [4] “A framework for supporting network continuity in vehicular ipv6 communications - scientific figure on researchgate.” https://www.researchgate.net/figure/ISO-ETSI-reference-communication-stack_fig1_260536157. [Citado nas páginas vii e 12]
- [5] “Etsi en 302 637-2 v1.3.1.” https://www.etsi.org/deliver/etsi_en/302600_302699/30263702/01.03.01_30/en_30263702v010301v.pdf. [Citado nas páginas vii, 13 e 27]
- [6] “Etsi en 302 637-3 v1.2.1.” https://www.etsi.org/deliver/etsi_en/302600_302699/30263703/01.02.01_30/en_30263703v010201v.pdf. [Citado nas páginas vii, 15 e 46]
- [7] B. Y. Yacheur, T. Ahmed, and M. Mosbah, “Analysis and comparison of ieee 802.11p and ieee 802.11bd,” (Cham), p. 55–65, 2020. [Citado nas páginas vii e 22]
- [8] “Ros-master-node-topic.png.” <https://upload.wikimedia.org/wikipedia/commons/e/e7/ROS-master-node-topic.png>. [Citado nas páginas vii e 25]
- [9] “Development of an emg-controlled mobile robot - scientific figure on researchgate.” https://www.researchgate.net/figure/The-ROS-rqt-graph-showing-various-nodes-of-the-application-and-the-topics-they-use_fig1_326215216. [Citado nas páginas vii e 26]

- [10] B. Vieira, R. Severino, E. V. Filho, A. Koubaa, and E. Tovar, “Copadrive - a realistic simulation framework for cooperative autonomous driving applications,” in *2019 IEEE International Conference on Connected Vehicles and Expo (ICCVE)*, pp. 1–6, 2019. [Citado nas páginas vii, 26 e 29]
- [11] “Platooningbackfinalnotype.jpg.” https://upload.wikimedia.org/wikipedia/commons/3/3a/Platooning_Back_022414_Final_noTpye.jpg. [Citado nas páginas vii e 34]
- [12] “Intersection.” https://www.freepik.com/free-vector/aerial-scene-intersection_5361164.htm#page=1&query=way%20intersection&position=2. [Citado nas páginas viii e 37]
- [13] F. Arena, G. Pau, and A. Severino, “A review on iee 802.11p for intelligent transportation systems,” *Journal of Sensor and Actuator Networks*, vol. 9, no. 2, p. 22, 2020. [Citado nas páginas viii, 10 e 51]
- [14] T. Renzler, M. Stolz, and D. Watzenig, *Decentralized Dynamic Platooning Architecture with V2V Communication Tested in Omnet++*. PhD thesis, IEEE, Austria, 2019. [Citado nas páginas 2 e 33]
- [15] R. M. P. Pacheco, “Evaluation of vehicle to everything environments: C-its simulation package,” Master’s thesis, Lisboa, Portugal, 2019. [Citado na página 5]
- [16] W. Nawapom, F. Bai, and M. Priyantha, “Broadcasting in vanet,” (Anchorage, AK), pp. 7–12, 2007. [Citado na página 5]
- [17] M. Conti and S. Giordano, “Mobile ad hoc networking: milestones, challenges, and new research directions,” *IEEE Communications Magazine*, vol. 52, no. 1, pp. 85–96, 2014. [Citado na página 6]
- [18] Y. Wang, A. Ahmed, B. Krishnamachari, and K. Psounis, “Ieee 802.11p performance evaluation and protocol enhancement,” (Columbus, OH), pp. 317–322, 2008. [Citado na página 8]
- [19] Y. Li, W. Chen, K. Zhang, T. Zheng, and H. Feng, “Dsrc based vehicular platoon control considering the realistic v2v/v2i communications,” (Chongqing, China), pp. 7585–7590, 2017. [Citado na página 8]
- [20] A. M. S. Abdelgader and W. Lenan, “The physical layer of the iee 802.11p wave communication standard: The specifications and challenges,” p. 8, 2014. [Citado na página 8]
- [21] J. A., A.-K. S., and D. A., “Performance evaluation of iee 1609 wave and iee 802.11p for vehicular communications,” (Poznan, Poland), pp. 1–5, 2012. [Citado na página 8]

- [22] S. Grafling, P. Mahonen, and J. Riihijarvi, “Performance evaluation of iee 1609 wave and iee 802.11p for vehicular communications,” (Jeju Island, Korea (South)), pp. 344–348, 2010. [Citado na página 8]
- [23] S. Eichler, “Performance evaluation of the iee 802.11p wave communication standard,” (Baltimore, MD, USA), pp. 2199–2203, 2007. [Citado na página 8]
- [24] L.-W. Chen, Y.-C. Tseng, and K.-Z. Syue, “Surveillance on-the-road: Vehicular tracking and reporting by v2v communications,” *IEEE Communications Magazine*, vol. 67, pp. 154–163, 2014. [Citado na página 9]
- [25] K. Bilstrup, E. Uhlemann, E. G. Strom, and U. Bilstrup, “Evaluation of the iee 802.11p mac method for vehicle-to-vehicle communication,” (Calgary, Canada), pp. 1–5, 2008. [Citado na página 10]
- [26] S. Benkirane and M. Benaziz, “Performance evaluation of iee 802.11p and iee 802.16e for vehicular ad hoc networks using simulation tools,” (Marrakech), pp. 573–577, 2018. [Citado na página 10]
- [27] A. Bazzi, B. M. Masini, A. Zanella, and I. Thibault, “On the performance of iee 802.11p and lte-v2v for the cooperative awareness of connected vehicles,” *IEEE Transactions on Vehicular Technology*, vol. 66, no. 11, pp. 10419–10432, 2017. [Citado na página 10]
- [28] M. Arndt, “Etsi - automotive intelligent transport | intelligent transport systems (its).” <https://www.etsi.org/technologies/automotive-intelligent-transport>. [Citado na página 10]
- [29] “Iee 1609 - family of standards for wireless access in vehicular environments (wave).” <https://www.standards.its.dot.gov/Factsheets/Factsheet/80>. [Citado na página 10]
- [30] J. Larmouth, *ASN.1 Complete*. Morgan Kaufmann, 2000. [Citado na página 14]
- [31] I. K. Virdaus, M. Kang, S. Shin, C. G. Lee, and J.-Y. Pyun, “A counting-based broadcast model of emergency message dissemination in vanets,” in *2017 Ninth International Conference on Ubiquitous and Future Networks (ICUFN)*, pp. 927–930, 2017. [Citado na página 16]
- [32] S. Kuhlmorgen, I. Llatser, A. Festag, and G. Fettweis, “Performance evaluation of etsi geonetworking for vehicular ad hoc networks,” in *2015 IEEE 81st Vehicular Technology Conference (VTC Spring)*, pp. 1–6, 2015. [Citado na página 16]

- [33] N. Lyamin, A. Vinel, D. Smely, and B. Bellalta, “Etsi dcc: Decentralized congestion control in c-its,” *IEEE Communications Magazine*, vol. 56, no. 12, pp. 112–118, 2018. [Citado na página 17]
- [34] “Omnet++.” <https://omnetpp.org/>. [Citado na página 19]
- [35] A. Varga, “Using the omnet++ discrete event simulation system in education,” *IEEE Transactions on Education*, vol. 42, no. 4, pp. 11 pp.–, 1999. [Citado na página 20]
- [36] “Artery.” <http://artery.v2x-research.eu/>. [Citado na página 21]
- [37] R. Riebl, H.-J. Günther, C. Facchi, and L. Wolf, “Artery: Extending veins for vanet applications,” in *2015 International Conference on Models and Technologies for Intelligent Transportation Systems (MT-ITS)*, pp. 450–456, 2015. [Citado na página 22]
- [38] “Veins - the open source vehicular network simulation framework.” <https://veins.car2x.org/>. [Citado na página 22]
- [39] R. Riebl, C. Obermaier, S. Neumeier, and C. Facchi, “Vanetza: Boosting research on inter-vehicle communication,” *Proceedings of the 5th GI/ITG KuVS Fachgespräch Inter-Vehicle Communication (FG-IVC 2017)*, pp. 37–40, 2017. [Citado na página 22]
- [40] “Gazebo simulator.” <http://gazebo.org/>. [Citado na página 23]
- [41] “Open dynamics engine.” <http://www.ode.org/>. [Citado na página 23]
- [42] I. Bzhikhatlov and S. Perepelkina, “Research of robot model behaviour depending on model parameters using physic engines bullet physics and ode,” in *2017 International Conference on Industrial Engineering, Applications and Manufacturing (ICIEAM)*, pp. 1–4, 2017. [Citado na página 23]
- [43] B. Vieira, R. Severino, A. Koubaa, and E. Tovar, “Towards a realistic simulation framework for vehicular platooning applications,” in *2019 IEEE 22nd International Symposium on Real-Time Distributed Computing (ISORC)*, pp. 93–94, 2019. [Citado na página 23]
- [44] “Ros melodic morenia.” <http://wiki.ros.org/melodic>. [Citado na página 23]
- [45] “Ros.” <https://www.ros.org/>. [Citado na página 24]
- [46] “Node.js.” <https://nodejs.org/en/>. [Citado na página 25]
- [47] C. Bergenheim, S. Shladover, and E. Coelingh, “Overview of platooning systems,” *Proceedings of the 19th ITS World Congress*, p. 8, 2012. [Citado na página 29]

-
- [48] T. Linget, “C-v2x use cases volume ii: Examples and service level requirements,” (München, Germany), p. 113, 2020. [Citado na página 33]
 - [49] A. Petrillo, A. Pescapé, and S. Santini, “A collaborative approach for improving the security of vehicular scenarios: The case of platooning,” vol. 122, pp. 59–75, 2018. [Citado na página 34]
 - [50] S. Thormann, A. Schirrer, and S. Jakubek, “Safe and efficient cooperative platooning,” *IEEE Transactions on Intelligent Transportation Systems*, pp. 1–13, 2020. [Citado na página 34]