CISTER

Research Centre in
Real-Time & Embedded
Computing Systems

BEng Thesis

Simulação de comunicação inter-veicular sobre LTE e 802.11p

Tiago Cerqueira

CISTER-TR-161202

2015/10/28

Simulação de comunicação inter-veicular sobre LTE e 802.11p

Tiago Cerqueira

*CISTER Research Centre

Polytechnic Institute of Porto (ISEP-IPP)

Rua Dr. António Bernardino de Almeida, 431

4200-072 Porto

Portugal

Tel.: +351.22.8340509, Fax: +351.22.8321159

E-mail: 1090678@isep.ipp.pt

http://www.cister.isep.ipp.pt

Abstract

Last years have witnessed the appearing of the vehicular network concept, whose applications can potentially raise road safety, reduce pollution and manage car fleets efficiently. Two communication technologies stand out as support for this kind of networks, the Long Term Evolution (LTE) and the 802.11p/Wireless Access in Vehicular Environments (WAVE).Given the particular characteristics of this kind of networks, most studies are executed by means of network simulators and mobility models, the latter to simulate the impact of mobility on communication activities. This work describes the implementation of two different algorithms for data dissemination, based on LTE and 802.11p respectively, and the implementation of a new mobility model for the ns-3 network simulator.The work comprises the design of two different algorithms for data dissemination based on the publish/subscribe paradigm, the development of applications implenting the algorithms in the ns-3 framework, the development of a mobility model that exploits an external travel-planning API such as Google Maps to produce the mobility traces for nodes. The applications described in this work allow to analyze the performance of data dissemination depending on which algorithms are used, the density of nodes in the network, the route followed by the nodes, and other parameters. The mobility model developed in the context of this thesis represents a contribution by itself, since it provides network researchers with realistic node mobility, without forcing them to resort to complex road traffic simulator. The module has been included into the release of the ns-3 network simulator platform.

Simulação de comunicação inter-veicular sobre LTE e 802.11p

CISTER – Centro de Investigação em Sistemas Confiáveis e de Tempo Real

2014 / 2015

1090678 Tiago Cerqueira

Simulação de comunicação inter-veicular sobre LTE e 802.11p

CISTER – Centro de Investigação em Sistemas Confiáveis e de Tempo Real

2014/ 2015

1090678 Tiago Cerqueira

Licenciatura em Engenharia Informática

Outubro de 2015

Orientador ISEP: **Professor Doutor Luís Lino Ferreira**

Supervisor Externo: **Professor Doutor Michele Albano**

Aos meus pais, à minha namorada e amigos

Agradecimentos

Em primeiro lugar gostaria de endereçar um forte agradecimento ao Professor Doutor Luís Lino Ferreira e ao Professor Doutor Michele Albano, por me terem dado todo o apoio ao longo deste projeto e pela dedicação e disponibilidade demonstradas.

De seguida quero agradecer também aos meus pais pelo apoio prestado ao longo de todos estes anos. Gostaria, ainda, de endereçar um agradecimento especial à minha namorada, por toda a dedicação e apoio ao longo desta etapa tão importante da minha vida e pela paciência nas fases mais difíceis.

Por último, gostaria de agradecer ao Fábio Oliveira, companheiro de estágio e de licenciatura, por toda a amizade ao longo destes anos.

Resumo

Nos últimos anos temos assistido ao aparecimento do conceito de redes veiculares, cujas aplicações tem o potencial de permitir um aumento da segurança rodoviária, redução de emissões e gestão de frotas eficiente, entre outras. Neste contexto, destacam-se duas tecnologias de comunicação para este tipo de redes, o *Long Term Evolution* (LTE) e o 802.11p/*Wireless Access in Vehicular Environments* (WAVE). Devido às características próprias deste tipo de redes, atualmente, a grande maioria dos estudos é realizada com recurso a simuladores de redes e, de forma a simular o impacto da mobilidade dos nós na rede, é necessária a utilização de modelos de mobilidade.

Neste trabalho é apresentada a implementação de dois algoritmos distintos de disseminação de dados, recorrendo a LTE e 802.11p, bem como a implementação de um novo modelo de mobilidade para o simulador de redes ns-3. O trabalho compreende o desenvolvimento de aplicações, no ns-3, que simulem a disseminação de dados recebidos, por parte de veículos com capacidades de comunicação com a infraestrutura, com a restante rede veicular. A implementação define dois algoritmos distintos, segundo o paradigma *publish/subscribe*, para a disseminação dos dados. Adicionalmente, foi ainda desenvolvido um módulo de mobilidade para o ns-3 que, recorrendo a APIs externas de planeamento de viagens, tais como as do Google Maps, é capaz de gerar mobilidade realística para os nós do simulador.

As aplicações descritas neste relatório permitem a análise da disseminação de dados numa rede veicular, recorrendo a dois algoritmos distintos, bem como o estudo do impacto das diferentes densidades de nós na rede. A implementação do modelo de mobilidade descrito neste relatório, por sua vez, torna-se um contributo para os investigadores da área de redes veiculares ou metropolitanas, uma vez que permite a simulação de mobilidade realística com facilidade, sem recurso a simuladores veiculares externos que, devido à sua complexidade e propósito, podem revelar-se de difícil configuração e compreensão. O módulo de mobilidade descrito encontra-se atualmente em fase de revisão de código, com vista à inclusão no simulador ns-3.

Palavras-chave (Tema): **VANET, ns-3, modelos de mobilidade, algoritmos de disseminação de dados**

Palavras-chave (Tecnologias): **Google Maps APIs, LTE, WAVE, SUMO**

Índice

Índice de Figuras

Índice de Tabelas

Notação e Glossário

CISTER	*Research Center in Real-Time & Embedded Computer Systems*
V2V	*Vehicle to Vehicle*
V2I	*Vehicle to Infrastructure*
VANET	*Vehicular ad hoc network*
LTE	*Long Term Evolution*
WAVE	*Wireless Access in Vehicular Environments*
IEEE	*Institute of Electrical and Electronics Engineers*
DSRC	*Digital Short Radio Communications*
ODFM	*Orthogonal Division Frequency Multiplexing*
LLC	*Logical Link Control*
BSS	*Basic Service Set*
WSMP	*Wave Short Message Protocol*
WBSS	*Wave Basic Service Set*
CSMA/CA	*Colision Sense Multiple Access/Colision Avoidance*
CCH	*Control Channel*
SCH	*Service Channel*
BSSID	*Basic Service Set Identification*
OCB	*Outside the Context of a Basic Service Set*
MSDU	*Mac Service Data Unit*
EDCA	*Enhanced Distributed Channel Access*
WME	*Wave Management Entity*
OBU	*On Board Unit*
RSU	*Road Side Unit*

UMTS	*Universal Mobile Telecommunications System*
3GPP	*Third Generation Partnership Project*
PS	*Packet Switched*
GSM	*Global System for Mobile communications*
CS	*Circuit Switched*
UE	*User Equipment*
PDN	*Packet Data Network*
SAE	*System Architecture Evolution*
EPC	*Evolved Packet Core*
EPS	*Evolved Packet System*
MT	*Mobile termination*
TE	*Termination Equipment*
eNB	*Evolved Node B*
P-GW	*Packet Data Network Gateway*
S-SW	*Serving Gateway*
MME	*Mobility Management Entity*
NAS	*Non-Access Stratum*
GTP	*GPRS Tunneling Protocol*
PDCP	*Packet Data Convergence Protocol*
RLC	*Radio Link Control*
MAC	*Medium Access Control*
PLMN	*Public Land Mobile Network*
DCE	*Direct Code Execution*
RB	*Resource Block*
MANET	*Mobile Ad-hoc Networks*
PSID	*Provider Service Identifier*

SUMO	*Simulation of Urban Mobility*
API	*Application Programming Interface*
XML	*eXtensible Markup Language*
ns-3	*Network Simulation 3*
RM	*Resource Manager*
EPOCH	*Os segundos desde o dia 1 de Janeiro de 1970*
AODV	*Ad hoc On Demand Distance Vector*
OLSR	*Optimized Link State Routing Protocol*
DSR	*Dynamic Source Routing*
DSDV	*Destination-Sequence Distance Vector*

1 Introdução

Este capítulo vai apresentar, sucintamente, o contexto e enquadramento do projecto "Simulação de comunicação inter-veicular sobre LTE e 802.11p". O capítulo irá, também, abordar as reuniões efetuadas e o planeamento do projeto.

1.1 Enquadramento

Este projeto realiza-se no âmbito da cadeira de Projeto/Estágio do 2º Semestre da Licenciatura em Engenharia Informática do Instituto Superior de Engenharia do Porto, que visa, acima de tudo, oferecer aos alunos uma aproximação ao mercado de trabalho na área de estudo do curso, permitindo ao aluno aplicar os conhecimentos adquiridos ao longo do mesmo e, desta forma, contribuir para o seu desenvolvimento pessoal e profissional.

O projeto "Simulação de comunicação inter-veicular sobre LTE e 802.11p", apresentado neste relatório, foi desenvolvido no contexto do projeto "CarCoDe – Platform for Smart Car to Car Content Delivery", um projeto de investigação do CISTER – Centro de Investigação em Sistemas Confiáveis e de Tempo Real.

1.2 Apresentação do projeto/estágio

Atualmente, o número de dispositivos que um ser humano possui com capacidade de rede é muito superior ao de há apenas uma década atrás, sendo que a tendência é de um aumento substancial durante os próximos anos.

Aplicar capacidades de rede a um veículo é um passo lógico e que poderá trazer inúmeras vantagens, desde a segurança ao conforto.

Neste contexto, o CISTER integra-se num consórcio europeu de empresas, universidades e centros de investigação, com o objetivo de introduzir *smart car-to-car content delivery*. O principal objetivo deste projeto é o de criar uma plataforma de software automóvel, independente do sistema, capaz de trocar informação em tempo real [1]. São exemplos práticos deste tipo de tecnologia o reporte de avarias no veículo em tempo real, o aviso de que se encontra próximo de um veículo de emergência e informação de posicionamento e estado de frotas de veículos, entre muitos outros.

A abordagem do projeto CarCoDe é a de reduzir os custos de comunicação e de redundância de informação, na partilha de conteúdos em tempo real. Esta abordagem inclui o melhoramento de redes intra-veiculares, de aplicações de comunicação *Vehicle to Vehicle* (V2V) e *Vehicle to Infrastructure* (V2I) e dos sensores existentes nos veículos atuais. É também fulcral identificar tecnologias de comunicação apropriadas [1].

É no sentido de analisar e identificar as tecnologias de comunicação apropriadas a *Vehicular ad hoc networks* (VANETs) que surge a motivação para este estágio curricular. O principal objetivo do estágio curricular é avaliar o desempenho de duas tecnologias de comunicação distintas, *Long Term Evolution* (LTE) e *802.11p/Wireless Access in Vehicular Environments* (WAVE), em diferentes cenários simulados.

É também objetivo do estágio o desenvolvimento de um módulo de mobilidade para o simulador de rede *open-source*, o *network simulator 3* (ns-3), de forma a auxiliar o estudo do desempenho da VANET.

1.2.1 Planeamento de projeto

O planeamento deste projeto compreendeu, essencialmente, três fases: análise, implementação e obtenção e análise de resultados. Inicialmente, foi estimado que a conclusão deste projeto seria a 19 de Setembro de 2014, contudo, devido a problemas alheios ao CISTER, só foi possível conhecer os requisitos necessários para a implementação da aplicação de disseminação de dados no final do mês de Setembro. Como tal, e uma vez que existia ainda

algum trabalho a realizar de forma a submeter um artigo sobre o módulo de mobilidade desenvolvido, foi necessário prolongar o prazo de entrega. Uma vez que este prazo foi prolongado, foram também adicionados novos objetivos ao projeto de estágio, nomeadamente a elaboração de um segundo artigo, sobre as simulações de disseminação de dados efetuadas, uma versão melhorada, com novas funcionalidades, do módulo de mobilidade desenvolvido inicialmente, a implementação de um segundo algoritmo de disseminação de dados e, ainda, a participação na *Workshop on ns-3 2015* (WNS3), que necessitou de uma preparação prévia.

No sentido de responder adequadamente a todos os desafios, foi prolongada sucessivamente a data de entrega do projeto, sendo definidos novos objetivos.

A Figura 1 apresenta o diagrama de Gantt demonstrativo do planeamento final, que se encontra ampliado no Anexo 1.

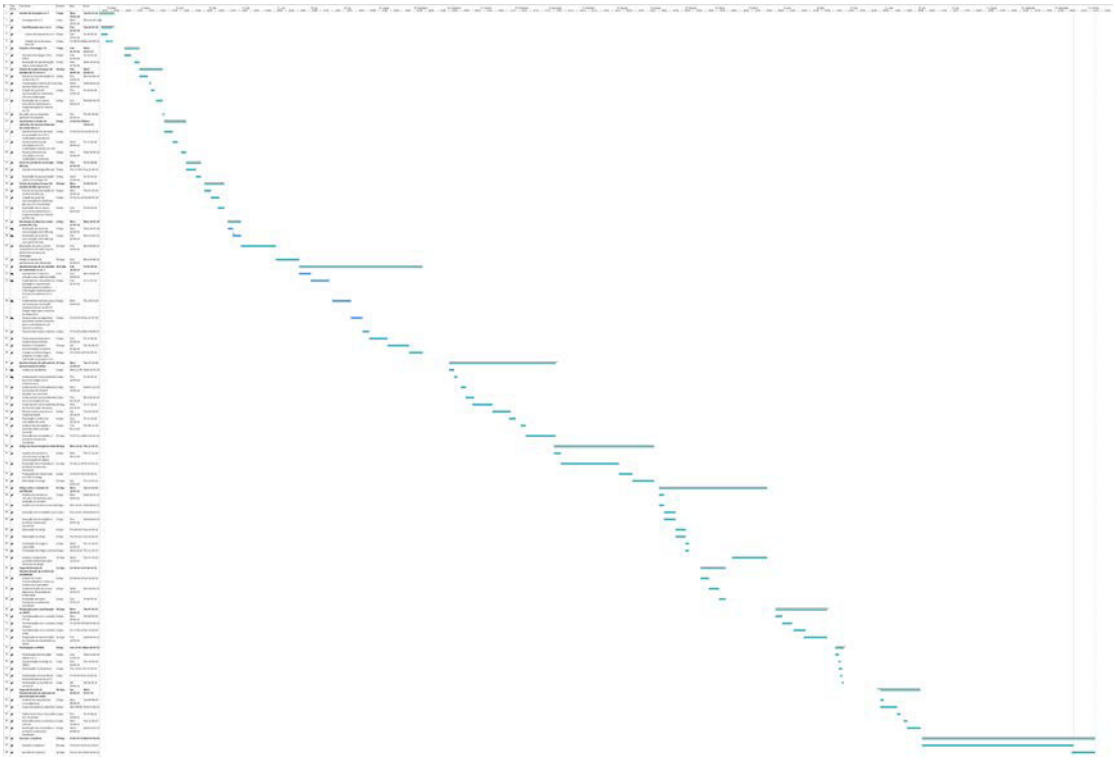


Figura 1 - Diagrama de Gantt do planeamento do projeto

1.2.2 Reuniões de acompanhamento

Data	Participantes	Local	Assunto
20/02/2014	Luís Ferreira e Michele Albano	CISTER	Progresso do tutorial e de configuração inicial do ns-3

Data	Participantes	Local	Assunto
27/02/2014	Luís Ferreira e Michele Albano	CISTER	Progresso do tutorial de ns-3 e início do estudo da tecnologia LTE. Elaboração e viabilidade do projeto a submeter ao *Google Summer of Code 2014*
5/3/2014	Luís Ferreira e Michele Albano	CISTER	Progresso do tutorial de ns-3 e início do estudo da tecnologia LTE. Elaboração e viabilidade do projeto a submeter ao *Google Summer of Code 2014* (GSOC)
7/3/2014	Luís Ferreira e Michele Albano	CISTER	Progresso do trabalho efetuado na proposta para o GSOC 2014
10/3/2014	Luís Ferreira e Michele Albano	CISTER	Progresso da apresentação sobre LTE
18/3/2014	Luís Ferreira, Michele Albano, César Teixeira e Fábio Oliveira	CISTER	Progresso do trabalho e breve apresentação sobre LTE.
20/3/2014	Luís Ferreira e Michele Albano	CISTER	Finalização e submissão da proposta ao GSOC 2014
27/3/2014	Michele Albano	CISTER (via Skype)	Progresso da simulação de um cenário simples com LTE e mobilidade. Progresso geral do trabalho
31/3/2014	Michele Albano	CISTER (via Skype)	Progresso do trabalho e discussão do estado da implementação do módulo de LTE no ns-3
1/4/2014	Luís Ferreira, Michele Albano e Luís Miguel Pinho	CISTER	Participação na reunião geral do projeto CarCoDe
2/4/2014	Luís Ferreira, Michele Albano e Luís Miguel Pinho	CISTER	Participação na reunião geral do projeto CarCoDe
7/4/2014	Luís Ferreira e Michele Albano	CISTER	Discussão sobre a transformação da informação obtida pelas APIs do Google Maps em mobilidade no ns-3
14/4/2014	Luís Ferreira e Michele Albano	CISTER	Discussão sobre o progresso de uma aplicação *proof-of-concept* para a obtenção de mobilidade, no ns-3, a partir dos dados obtidos pela API do Google Maps
17/4/2014	Luís Ferreira e Michele Albano	CISTER	Discussão sobre o estado da implementação atual do módulo de WAVE no ns-3

Data	Participantes	Local	Assunto
21/4/2014	Luís Ferreira e Michele Albano	CISTER	Implementação de simulação básica do protocolo WAVE
28/4/2014	Luís Ferreira e Michele Albano	CISTER	Implementação de simulação com recurso a LTE e WAVE.
30/4/2014	Luís Ferreira e Michele Albano	CISTER	Analise dos *e-mails* trocados com o criador do módulo WAVE
5/5/2014	Luís Ferreira e Michele Albano	CISTER	Progresso do trabalho em geral
6/5/2014	Luís Ferreira e Michele Albano	CISTER	Análise de *e-mails* trocados com o criador do módulo WAVE
8/5/2014	Michele Albano	EVOLEO	Reunião da equipa portuguesa do CarCoDe
19/5/2014	Luís Ferreira e Michele Albano	CISTER	Progresso do trabalho em geral
26/5/2014	Luís Ferreira e Michele Albano	CISTER	Progresso do trabalho em geral
27/5/2014	Luís Ferreira e Michele Albano	CISTER	Análise de *e-mails* trocados com o criador do módulo WAVE
2/6/2014	Luís Ferreira e Michele Albano	CISTER	Discussão de simulações WAVE usando IP
9/6/2014	Luís Ferreira e Michele Albano	CISTER	Análise do módulo de mobilidade a implementar
16/6/2014	Luís Ferreira e Michele Albano	CISTER	Progresso do desenvolvimento do módulo de mobilidade
23/6/2014	Luís Ferreira e Michele Albano	CISTER	Progresso do desenvolvimento do módulo de mobilidade
30/6/2014	Luís Ferreira e Michele Albano	CISTER	Finalização da primeira iteração do módulo
7/7/2014	Luís Ferreira e Michele Albano	CISTER	Discussão das funcionalidades a implementar na segunda iteração do módulo
14/7/2014	Luís Ferreira e Michele Albano	CISTER	Criação de testes unitários e documentação de classes
21/7/2014	Luís Ferreira e Michele Albano	CISTER	Finalização da segunda iteração do módulo
28/7/2014	Luís Ferreira e Michele Albano	CISTER	Criação de documentação do módulo
12/8/2014	Luís Ferreira e Michele Albano	CISTER	Integração do módulo criado com o simulador
18/8/2014	Luís Ferreira e Michele Albano	CISTER	Balanço do trabalho elaborado. Melhoria da documentação
1/9/2014	Luís Ferreira e Michele Albano	CISTER	Submissão da primeira versão do módulo para revisão
8/9/2014	Luís Ferreira e Michele Albano	CISTER	Balanço final do trabalho realizado, discussão de trabalho futuro no módulo.
20/9/2014	Luís Ferreira e Michele Albano	CISTER	Ponto de situação das capacidades do ns-3 para o projeto CarCoDe

Data	Participantes	Local	Assunto
6/10/2014	Luís Miguel Pinho e Michele Albano	CISTER	Discussão dos algoritmos de disseminação de dados
13/10/2014	Luís Ferreira e Michele Albano	CISTER	Progresso da implementação
20/10/2014	Luís Ferreira e Michele Albano	CISTER	Progresso dos testes efetuados. Discussão dos parâmetros de simulação a analisar
27/10/2014	Luís Ferreira e Michele Albano	CISTER	Finalização da aplicação de disseminação de dados. Realização de algumas simulações de teste
4/11/2014	Luís Ferreira e Michele Albano	CISTER	Discussão dos resultados de testes obtidos
6/11/2014	Luís Ferreira e Michele Albano	CISTER	Discussão das simulações a realizar para o projeto CarCoDe
14/11/2014	Luís Ferreira e Michele Albano	CISTER	Discussão dos resultados das simulações
18/11/2014	Luís Ferreira e Michele Albano	CISTER	Realização de simulações adicionais
20/11/2014	Luís Ferreira e Michele Albano	CISTER	Discussão dos resultados das simulações
3/12/2014	Luís Ferreira e Michele Albano	CISTER	Analise dos resultados obtidos
4/12/2014	Luís Ferreira e Michele Albano	CISTER	Discussão sobre a representação dos resultados de simulação no artigo para a WFCS
10/12/2014	Luís Ferreira e Michele Albano	CISTER	Discussão sobre os cenários a estudar para o artigo
16/12/2014	Luís Ferreira e Michele Albano	CISTER	Progresso das simulações
17/12/2014	Luís Ferreira e Michele Albano	CISTER	Discussão sobre a estrutura deste relatório
18/12/2014	Luís Ferreira e Michele Albano	CISTER	Discussão sobre o progresso das simulações
30/12/2014	Luís Ferreira e Michele Albano	CISTER	Discussão sobre o progresso das simulações
6/1/2015	Luís Ferreira e Michele Albano	CISTER	Discussão sobre o progresso das simulações
7/1/2015	Luís Ferreira e Michele Albano	CISTER	Discussão sobre o progresso das simulações
8/1/2015	Luís Ferreira e Michele Albano	CISTER	Discussão sobre o progresso das simulações
9/1/2015	Luís Ferreira e Michele Albano	CISTER	Discussão sobre a análise das simulações
12/1/2015	Luís Ferreira e Michele Albano	CISTER	Discussão sobre a análise das simulações
13/1/2015	Luís Ferreira e Michele Albano	CISTER	Discussão sobre a análise das simulações

Data	Participantes	Local	Assunto
15/1/2015	Luís Ferreira e Michele Albano	CISTER	Discussão sobre a análise das simulações
19/1/2015	Luís Ferreira e Michele Albano	CISTER	Discussão sobre os dados a inserir no artigo. Discussão sobre o progresso do artigo
22/1/2015	Luís Ferreira e Michele Albano	CISTER	Finalização do artigo *Data Dissemination by Extending Publish/Subscribe to Vehicular Environments*, submetido à WFCS 2015
26/1/2015	Luís Ferreira e Michele Albano	CISTER	Discussão sobre o artigo sobre o módulo de mobilidade, a submeter para a WNS3 2015
28/1/2015	Luís Ferreira e Michele Albano	CISTER	Estudo do trabalho existente relacionado com avaliação de modelos de mobilidade. Discussão sobre os dados a incluir no artigo
4/2/2015	Luís Ferreira e Michele Albano	CISTER	Discussão sobre o progresso do artigo
10/2/2015	Luís Ferreira e Michele Albano	CISTER	Discussão sobre o progresso do artigo
11/2/2015	Luís Ferreira e Michele Albano	CISTER	Discussão sobre o progresso do artigo
12/2/2015	Luís Ferreira e Michele Albano	CISTER	Finalização do artigo *RoutesMobilityModel: easy realistic mobility simulation using external information services* para submissão à WNS3 2015
20/2/2015	Luís Ferreira e Michele Albano	CISTER	Análise de novas funcionalidades a inserir no módulo de mobilidade
26/2/2015	Luís Ferreira e Michele Albano	CISTER	Analise e implementação de novos algoritmos de geração de mobilidade automaticamente
27/2/2015	Luís Ferreira e Michele Albano	CISTER	Discussão dos resultados obtidos usando as novas funcionalidades. Discussão sobre os testes efetuados
11/3/2015	Luís Ferreira e Michele Albano	CISTER	Analise das respostas obtidas pelos revisores do artigo
17/3/2015	Luís Ferreira e Michele Albano	CISTER	Discussão sobre as respostas a enviar aos revisores.
23/3/2015	Luís Ferreira, Michele Albano, Fábio Oliveira, Ricardo Garibay-Martinez	CISTER	Simulação do protocolo FTT-SE no ns-3

Data	Participantes	Local	Assunto
31/3/2015	Luís Ferreira e Michele Albano	CISTER	Preparação para a apresentação do artigo sobre o módulo de mobilidade na WNS3
1/4/2015	Michele Albano e Fábio Oliveira	CISTER	Preparação para uma breve apresentação sobre o módulo de FTT-SE na WNS3
2/4/2015	Michele Albano e João Loureiro	CISTER	Preparação para uma breve apresentação sobre o módulo XDENSE na WNS3.
7/4/2015	Michele Albano	CISTER	Preparação para uma breve apresentação sobre o módulo GPSR na WNS3.
12/4/2015	Michele Albano e Fábio Oliveira	CISTER	Preparação para uma breve apresentação sobre o módulo de FTT-SE na WNS3
14/4/2015	Michele Albano e João Loureiro	CISTER	Preparação para uma breve apresentação sobre o módulo XDENSE na WNS3.
16/4/2015	Michele Albano	CISTER	Preparação para uma breve apresentação sobre o módulo GPSR na WNS3.
23/04/2015	Luís Ferreira e Michele Albano	CISTER	Discussão sobre a versão final da apresentação para a WNS3
7/5/2015	Luís Ferreira e Michele Albano	CISTER	Preparação final para a apresentação do artigo na WNS3
12/5/2015	Michele Albano	Centre Tecnològic Telecomunicacions Catalunya (via Skype)	Discussão sobre o progresso da WNS3
14/5/2015	Michele Albano	Centre Tecnològic Telecomunicacions Catalunya	Discussão sobre o progresso da WNS3
4/6/2015	Luís Ferreira e Michele Albano	CISTER	Discussão de novo algoritmo de disseminação de dados a implementar
5/6/2015	Luís Ferreira e Michele Albano	CISTER	Analise dos requisitos do novo algoritmo
10/6/2015	Michele Albano	CISTER	Progresso da implementação
15/6/2015	Michele Albano	CISTER	Progresso da implementação. Discussão de resultados iniciais
17/6/2015	Luís Ferreira e Michele Albano	CISTER	Discussão sobre os cenários a simular
22/6/2015	Michele Albano	CISTER	Progresso das simulações
24/6/2015	Luís Ferreira e Michele Albano	CISTER	Analise dos resultados preliminares das simulações
30/6/2015	Luís Ferreira e Michele Albano	CISTER	Analise dos resultados finais das simulações
3/7/2015	Luís Ferreira e Michele Albano	CISTER	Discussão sobre o progresso do relatório

Data	Participantes	Local	Assunto
16/7/2015	Luís Ferreira e Michele Albano	CISTER	Discussão sobre o progresso do relatório
30/7/2015	Luís Ferreira e Michele Albano	CISTER	Discussão sobre o progresso do relatório

1.3 Tecnologias utilizadas

1.3.1 ns-3

O ns-3 é um *discrete event network simulator*, orientado para as áreas de investigação e educação. A arquitetura modular e o leque de tecnologias suportadas tornam-no ideal para o projeto realizado.

O ns-3 é distribuído sobre a licença *Gnu General Public Licence version 2*.

1.3.2 Netbeans IDE

No desenvolvimento deste projeto, foi utilizado o *Integrated Development Environment* (IDE) Netbeans. A escolha recaiu sobre este IDE, uma vez que existe suporte para o uso deste IDE com o ns-3.

1.3.3 Google Maps APIs

Foram utilizadas as APIs HTTP do Google Maps nomeadamente, a API *Directions* e a API *Places*, no desenvolvimento do módulo de mobilidade. Estas APIs foram escolhidas por serem completas e robustas, fornecendo todas as funcionalidades necessárias ao objetivo final proposto.

1.3.4 Doxygen

O Doxygen é uma ferramenta usada para a geração de documentação a partir de anotações em ficheiros de código fonte. O ns-3 faz uso desta ferramenta para a geração de documentação relativa às suas classes, pelo que foi necessário alguma familiarização com a mesma, de forma a documentar as classes implementadas.

1.3.5 Sphinx

O Sphinx é uma ferramenta capaz de gerar documentação bem organizada, facilmente. O projeto do ns-3 faz uso desta ferramenta na documentação dos seus módulos e, como tal, foi necessário alguma familiarização com a mesma, com vista a documentar detalhadamente o módulo de mobilidade desenvolvido.

1.3.6 cURLpp

O cURLpp é um *wrapper* C++ para a biblioteca libcURL, uma biblioteca de transferências livre e fácil de usar, com suporte para protocolos como HTTP, HTTPS e FTP, entre outros. Esta biblioteca adiciona uma camada *object oriented* à biblioteca libcURL.

1.3.7 GeographicLib

Esta biblioteca é composta por um conjunto de classes capazes de converter entre coordenadas geográficas, em variados formatos, e coordenadas cartesianas, entre outras funcionalidades.

1.3.8 Xerces-C++

A biblioteca Xerces-C++ fornece capacidades de *parsing*, geração, manipulação e validação de conteúdo XML, recorrendo a DOM (Document Object Model), SAX (Simple API for XML) e SAX2.

1.3.9 SAX2

O SAX2 fornece um algoritmo baseado em eventos, de *parse* de conteúdo XML, permitindo a leitura de elementos sequencialmente, sem a necessidade da leitura do documento completo, como acontece com o DOM.

1.3.10 Git

O Git é uma ferramenta de controlo de versões gratuita e de código fonte aberto, que foi usada no âmbito deste projeto, de forma a manter um arquivo do trabalho realizado. Foram criados dois repositórios Git no decurso deste projeto, um para manter o arquivo do trabalho do projeto de estágio e outro contendo uma versão costumizada do ns-3, com o modelo de mobilidade desenvolvido, disponível ao público, em geral.

1.4 Apresentação da organização

O Centro de Investigação em Sistemas Confiáveis e de Tempo Real (CISTER) é uma unidade de investigação Portuguesa de referência, baseada no Instituto Superior de Engenharia do Porto (ISEP) do Politécnico do Porto (IPP). O Centro foca a sua atividade de investigação na análise, projeto e implementação de sistemas de computadores embebidos e de tempo-real, sendo um dos líderes mundiais na investigação em diversos tópicos dentro das áreas das redes de sensores sem fio, das plataformas *multi-core* embebidas ou de *software* de tempo- real.

Desde que foi criado (em 1997), o CISTER cresceu até se tornar a unidade mais proeminente do ISEP, sendo o único Centro de I&D Português nas áreas da Engenharia Eletrotécnica e

Informática a obter consecutivamente a avaliação de Excelente nos últimos dois processos de avaliação de I&D em Portugal, realizada por painéis internacionais. O Centro participa consistentemente em projetos de Investigação, Desenvolvimento e Inovação (I&D&I) nacionais e internacionais, com parceiros como a Portugal Telecom, a Critical Software, a ISA, a Thales, a EADS, a Infineon, a SAP, a Schneider Electric ou a Embraer.

1.5 Contributos deste trabalho

1.5.1 RoutesMobilityModel

Até ao momento, o ns-3 possuía apenas modelos de mobilidade sintéticos nativos. Estes modelos, embora fáceis de executar, são demasiado irrealistas para estudos de impacto da mobilidade dos nós nas comunicações. A implementação de um gerador de mobilidade realística no ns-3 permite, a investigadores interessados em redes caracterizadas por elevada mobilidade de nós, estudar o impacto dessa mobilidade nas comunicações. Até ao momento, a geração de mobilidade realística tinha que ser, forçosamente, efetuada com recurso a outro *software*, nomeadamente simuladores de mobilidade urbana, como o SUMO. Estes simuladores, pela sua complexidade, exigiam que o utilizador estivesse também familiarizado com o mesmo, de forma a obter mobilidade complexa, com movimentos realísticos, para os seus nós.

O RoutesMobilityModel veio preencher a lacuna existente, entre os modelos de mobilidade fáceis de usar, mas irrealistas (modelos sintéticos), e modelos de mobilidade realísticos, mas com um elevado grau de dificuldade de utilização. Este novo módulo permite que o utilizador gere, com relativa facilidade, mobilidade realística e complexa, para todos os nós que pretende simular. O módulo possui, assim, a capacidade de simular o movimento de veículos sobre uma rede de estradas real, o movimento pedestre e o movimento pedestre com recurso a transportes públicos.

Desta forma, o RoutesMobilityModel, tem o potencial de permitir a investigadores de redes veiculares, por exemplo, a execução de simulações com mobilidade realística, sem a necessidade de recorrer a ferramentas alheias ao ns-3.

São, ainda, contributos deste trabalho, a análise das simulações efetuadas com recurso a este modelo de mobilidade, que permitiram validar o mesmo.

1.5.2 Aplicação CarCoDe

A aplicação desenvolvida teve, por objetivo principal, responder aos requisitos propostos pelo consórcio do projeto. Desta forma, a aplicação serve apenas para o estudo efetuado.

São, ainda, contributos deste trabalho, a análise efetuada às simulações, que permitem, por exemplo, determinar a concentração ideal de veículos para uma comunicação com sucesso em toda a rede veicular.

1.6 Organização do relatório

Este relatório encontra-se dividido em 9 capítulos principais, organizados da seguinte forma:

Capítulo 1 – Introdução: Neste capítulo é realizada uma apresentação e enquadramento do projeto de estágio, bem como a apresentação da organização e os contributos deste trabalho

Capítulo 2 – *Vehicular ad hoc networks*: Neste capitulo é apresentado o conceito de redes veiculares, que permite enquadrar o projeto desenvolvido.

Capitulo 3 - IEEE 802.11p/*Wireless Access in Vehicular Environments* (WAVE): Neste capitulo é apresentado o protocolo 802.11p/WAVE, componente essencial ao desenvolvimento deste projeto.

Capitulo 4 – Long Term Evolution (LTE): Este capítulo apresenta o protocolo LTE, componente essencial ao desenvolvimento deste projeto.

Capitulo 5 – Network Simulator 3 (ns-3): Este capítulo apresenta os aspetos gerais do simulador ns-3, bem como as componentes deste simulador, essenciais ao desenvolvimento do projeto.

Capitulo 6 – Serviços do Google Maps: Neste capítulo são descritas, em detalhe, as APIs do Google Maps usadas no âmbito deste projeto.

Capitulo 7 – Descrição Técnica: Neste capítulo são apresentados os requisitos do trabalho a realizar, bem como a descrição da implementação efetuada no âmbito deste projeto.

Capitulo 8 – Resultados e validação: Este capitulo descreve as experiencias realizadas, recorrendo ao trabalho realizado no âmbito deste projeto, bem como a analise dos resultados obtidos.

Capitulo 9 – Conclusão: Neste capítulo são apresentadas as conclusões finais do trabalho realizado, mencionando os objetivos concretizados, o trabalho futuro e a apreciação final ao projeto.

2 Vehicular ad hoc networks

A área de redes veiculares tem aplicações variadas, desde segurança veicular a serviços de *infotainment*. Algumas aplicações de redes veiculares estão já em uso, tais como sistemas de pagamento automático de portagens e sistemas de assistência à condução. No entanto é necessária uma abordagem que permita aos veículos comunicarem com dispositivos remotos (*V2I)* e com outros veículos (*V2V*).

O ambiente de comunicação veicular, contudo, apresenta desafios, uma vez que é preciso ter em conta a velocidade dos nós, a sua elevada mobilidade e consequentes mudanças de topologia da rede, bem como a elevada taxa de erros nas comunicações. É também importante considerar que a maioria das comunicações veiculares só serão úteis se comunicadas em tempo-real.[2]

2.1.1 Aplicações

A integração de interfaces de rede, recetor GPS e variados sensores com informação acerca do estado do veículo, com um computador, permitem construir poderosos sistemas de segurança veicular [3]. A informação recolhida permite, ainda, aplicações nas áreas de gestão e otimização de trânsito e frotas de veículos, bem como aplicações de *infotainment*. Desta

forma, as aplicações de VANETs são normalmente classificadas como aplicações de segurança rodoviária, gestão de trânsito e frotas de veículos e aplicações de *infotainment.*[2].

2.1.1.1 Segurança rodoviária

Esta categoria de aplicações foca-se na diminuição da probabilidade de acidentes e de perda de vidas humanas.

Uma percentagem significativa dos acidentes em todo o mundo envolvem cruzamentos e choques frontais, laterais e traseiros [2]. Neste contexto, aplicações de segurança rodoviária capazes de partilhar informação relativa à posição do veículo, velocidade e orientação, têm o potencial para reduzir acidentes.

O veículo poderá, também, fornecer informações sobre o estado da estrada que está a percorrer que, em conjugação com programas de navegação GPS, poderá informar outros veículos que um dado troço da estrada contem óleo ou outro fluido perigoso, por exemplo, redirecionando os outros veículos nas proximidades, se possível[3].

Diferentes sensores no veículo podem, também, colher informação vital sobre o estado do veículo, capacidade de travagem, velocidade atual, proximidade relativa de outros veículos, etc. A partilha desta informação poderá servir para avisar os condutores de que um veículo vizinho se encontra com uma avaria nos travões, por exemplo. Também é possível a um veículo detetar e alertar para a existência um veículo de emergência nas imediações antes que o condutor se aperceba, através dos sinais sonoros e luminosos tradicionais, do mesmo.

Alguns comportamentos humanos de risco podem também ser mitigados, uma vez que a colheita e partilha de informações dos sensores dos veículos permite detetar alguns desses comportamentos, tais como ultrapassagens agressivas, colisões devido a desacelerações repentinas, colisões devidas à mudança de faixa de rodagem, entre outras [3].

2.1.1.2 Gestão de trânsito e de frotas de veículos

Uma gestão eficiente do trânsito numa área metropolitana é essencial ao bom funcionamento da mesma ao mesmo tempo que ajuda a reduzir as emissões de dióxido de carbono para a atmosfera.

Estas aplicações focam-se na coordenação e otimização do trânsito, oferecendo informação acerca do trânsito num dado local. Esta informação permite reduzir ou, pelo menos, atenuar os engarrafamentos, oferecendo aos condutores, alternativas menos congestionadas.

As redes veiculares têm, também, o potencial para realizar navegação cooperativa como, por exemplo, *car-platooning*. Este tipo de navegação permite aumentar a segurança rodoviária,

removendo a problemática de mudanças de faixa de rodagem e de alterações de velocidade, enquanto ajuda a reduzir os congestionamentos, reduzindo, com segurança, a distância entre os carros.[4]

A gestão de frotas é também uma área que beneficiará do uso generalizado de VANETs, informando em tempo real a localização de todos os veículos da frota, otimizando rotas de distribuição, etc. Estas informações tem o potencial de economizar recursos monetários e empresariais.

2.1.1.3 Aplicações de *infotainment*

Este tipo de aplicações é responsável por transmitir, aos ocupantes de um veículo, informações relativas a locais de interesse nas proximidades do veículo, *streaming* de conteúdos audiovisuais, permitir, aos passageiros, o uso de jogos *online*, etc. [2]

2.1.2 Desafios à implementação de *VANETs*

Uma rede veicular apresenta características únicas, tais como:

- Mudanças rápidas de topologia.
- Autonomia energética virtualmente ilimitada.
- Densidade de rede variável e altamente dinâmica
- O condutor, ao reagir à informação apresentada pela rede, pode provocar mudanças na topologia da mesma.[3]

A investigação nesta área ocorre há já vários anos, de forma a combater as limitações deste tipo de redes. Neste contexto, a *Institute of Electrical and Electronics Engineers* (IEEE) tem vindo a desenvolver uma variante do *standard* 802.11, de forma a suportar comunicações V2V e V2I, capazes de funcionar a velocidades elevadas (até 200 km/hora) e através de distâncias até 1 quilómetro.[5]

Os esforços do IEEE resultaram no *standard* 802.11p, cuja camada física é baseada na camada física do *standard* 802.11a. Este *standard*, embora tenha sido desenvolvido para ser usado em redes veiculares, apresenta alguns problemas. Orientado para a comunicação V2V, o maior problema do 802.11p é o facto de ter sido desenvolvido para áreas de comunicação relativamente pequenas [6]. O 802.11p é também problemático a nível de escalabilidade da rede e não possui garantias de qualidade de serviço determinísticas[7],[8].

Nos últimos anos, tem-se vindo a assistir a um interesse crescente em outras tecnologias para cenários de comunicação inter-veicular, uma vez que o 802.11p não é capaz de manter uma

conexão contínua do tipo V2I sem uma implementação total, ou praticamente total, de infraestruturas de comunicação ao longo das rodovias[8]. O *Long Term Evolution* (LTE), em particular, tem vindo a ser considerada a tecnologia mais promissora, por possuir uma elevada largura de banda e baixa latência, beneficiando ainda de uma grande área de cobertura. Estas características fazem do LTE a tecnologia mais indicada para as aplicações do tipo *infotainment*, mas a sua capacidade para suportar aplicações com comunicações diretas, entre veículos, numa rede veicular, tais como domínios aplicativos de segurança rodoviária ou de eficiência de trânsito, esta ainda por ser provada[8].

Este trabalho pretende mostrar a eficiência de uma rede veicular, na disseminação de conteúdos de interesse geográfico, obtidos a partir de um servidor externo. A rede em estudo irá avaliar, em particular, a eficiência de uma rede contendo nós capazes de comunicação V2I e V2V, recorrendo a dispositivos LTE e WAVE.

3 IEEE 802.11p/*Wireless Access in Vehicular Environments* (WAVE)

Nesta secção é descrito em detalhe o protocolo IEEE 802.11p, bem como as extensões WAVE (*Wireless access in vehicular environments*).

Será também abordada a arquitetura WAVE e as características inerentes ao protocolo IEEE 802.11p.

3.1 Introdução

O protocolo IEEE 802.11p teve origem na autorização, por parte da *United States Federal Communication Commission,* para a alocação de 75 MHz do espectro *Dedicated Short Range Communications* (*DSRC*) a 5.9 GHz, para uso exclusivo de comunicações *V2V* e *V2I* [9].

O *standard* foi oficialmente aceite em Setembro de 2003, operando na frequência 5.85-5.925 GHz nos Estados Unidos e na frequência 5.855-5.905 GHz na Europa[10].

A camada física do protocolo é baseada na camada física do 802.11a, suportando *Orthogonal frequency-division multiplexing* (*ODFM*). No entanto, a camada física do 802.11p difere da camada física do 802.11a na medida em que o 802.11p permite, para além do funcionamento tradicional a 20 MHz, o funcionamento a 10 MHz, de forma a compensar o aumento do *delay*

spread em ambientes de comunicação veicular. O funcionamento a 10 MHz significa que a velocidade máxima é de 27 Mb/s, metade da velocidade tradicional do 802.11a [10].

Em 2004, um grupo de trabalho do IEEE deu início à criação da emenda ao standard IEEE 802.11, conhecido hoje como 802.11p. Outra equipa do IEEE deu início ao desenvolvimento de um conjunto de especificações, conhecidos como IEEE 1609.x, de forma a cobrir as camadas adicionais do protocolo. Neste momento, existem cinco especificações IEEE 1609.x (IEEE 1609.1, IEEE 1609.2, IEEE 1609.3, IEEE 1609.4, IEEE 1609.5) [11]. O *standard* 802.11p define a camada física e a camada de *Logical Link Control* (LLC), enquanto que as especificações IEEE 1609.x definem a camada MAC (*IEEE 1609.4*), a camada de rede (*IEEE 1609.3*), a camada de segurança (*IEEE 1609.2*), a camada de gestão de recursos (*IEEE 1609.1*) e a camada de gestão da comunicação (*IEEE 1609.5*)[10].

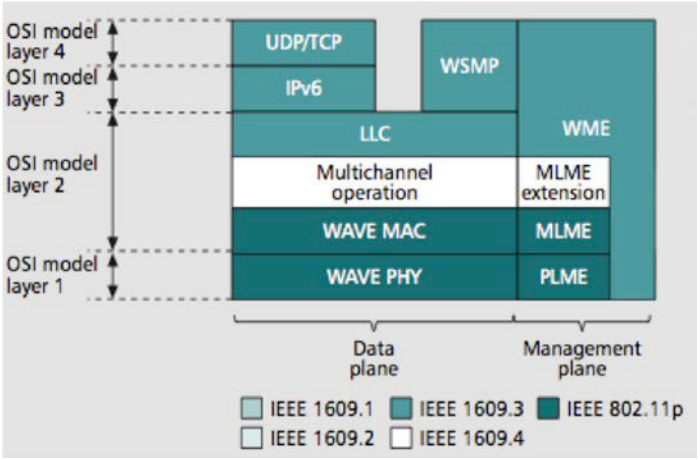


Figura 2 – Pilha protocolar WAVE [11]

Comparativamente ao *standard* IEEE 802.11, as alterações à camada MAC presentes no IEEE 802.11p focam-se na capacidade de comunicação dos nós, caracterizada por um tempo de inicialização menor, em comparação com as comunicações 802.11. As comunicações 802.11 necessitam que os diferentes nós mantenham grupos cooperativos (*Basic Service Set* [BSS]), para permitir a comunicação entre os nós[9], o 802.11p possui um BSS único, o *Wave Basic Service Set* (WAVE BSS, ou WBSS), de forma a permitir uma comunicação fiável com a infraestrutura sem o *overhead* inerente à gestão do WBSS. Na prática, isto traduz-se num melhor suporte em cenários de elevada mobilidade dos nós. O 802.11p suporta, assim, comunicações sobre IPv6 ou sobre o *Wave Short Message Protocol* (WSMP), sendo este último indicado para comunicações com restrições temporais. O WSMP e o WAVE BSS serão abordados numa próxima secção deste relatório.

3.2 Arquitetura WAVE

O sistema WAVE foi desenhado para permitir comunicações V2V e V2I, de forma a acomodar aplicações de segurança rodoviária, por exemplo, e aplicações de entretenimento. Existem, então, dois componentes essenciais num sistema WAVE: as *roadside units* (RSU) e as *On-board units* (OBU)[11], [12].

As RSU são equipamentos montados ao longo de uma rede de estradas, e visam permitir a comunicação dos veículos ao seu alcance, com a restante infraestrutura WAVE (V2I), desta forma permitindo que o veículo receba atualizações acerca do estado do trânsito nas vias em que vai circular, por exemplo[11].

As OBUs, por sua vez, são equipamento montado no interior do veículo, visando permitir a comunicação do mesmo com o exterior. Para este efeito, podem formar um WAVE BSS, para comunicações sem restrições temporais, entre OBUs (V2V) ou entre OBUs e RSUs (V2I). Podem também optar por comunicar em modo OCB, normalmente usado para comunicações curtas e urgentes[11].

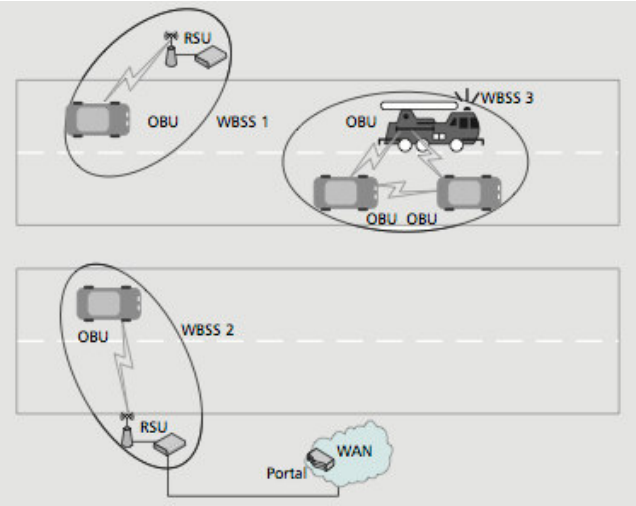


Figura 3 - Componentes de um sistema WAVE[11]

3.3 Camada Física

Ao nível da camada física foram realizadas apenas pequenas alterações, uma vez que é fundamental uma comunicação eficiente entre nós que se deslocam a elevadas velocidades.

Tal como referido anteriormente, a frequência de comunicação é de 10 MHz, o que representa metade da frequência habitual definida para o 802.11a. Uma vez que o standard 802.11 já define o uso de canais de 10 MHz, é apenas necessário aumentar para o dobro os parâmetros de *timing* do OFDM usados nas comunicações tradicionais sobre 802.11a, a 20 MHz[9].

Foram também realizadas alterações à *performance* do rádio, uma vez que a natureza da rede, com veículos a moverem-se muito próximo e a diferentes velocidades, é propícia a interferências entre os canais. O método mais indicado resolver este problema é a implementação de políticas de gestão de canal, que não estão definidas no standard 802.11.

O *standard* 802.11p faz, ainda, uso de *Colision Sense Multiple Access/Colision Avoidance* (CSMA/CA) com o objetivo de reduzir o número de colisões e de garantir uma utilização justa do canal de comunicação[13]. O 802.11p dispõem de um *Control Channel* (CCH) e seis *Service Channels* (SCH) [11], [14].

3.4 Camada *MAC*

A camada *MAC* descrita no standard 802.11p, tal como mencionado anteriormente, foca-se na diminuição do *overhead* da criação de *BSS*, uma vez que é impraticável a formação de uma associação a um *access point*, dada a elevada mobilidade dos nós e mudanças constantes de topologia.

Desta forma, foi necessário implementar um tipo diferente de *BSS*, o WAVE *BSS*, que colmata os problemas mencionados anteriormente. Numa operação típica, um nó emite um pedido (*beacon*) que, ao ser recebido por outra estação, desencadeia o envio da informação necessária para formar uma associação *ad-hoc* com o nó. Como tal, não é necessário esperar pelo envio periódico de um *beacon*, que caracteriza o funcionamento 802.11 tradicional. O método descrito traduz-se num *overhead* consideravelmente mais baixo para a formação de um WAVE *BSS*, mas descarta todo o processo de autenticação e associação, que terá que ser gerido pelas camadas superiores[9].

Enquanto este método pode ser adequado a comunicações com baixa prioridade, tais transferencia de informação de trânsito ou de música, o mesmo já não se pode afirmar de comunicações urgentes, tais como avisos de colisão eminente. Desta forma, um nó, mesmo após ter formado uma associação com um WAVE *BSS*, continua a ser capaz de comunicar com outros nós colocando, na *frame*, todos os bits correspondentes ao *Basic Service Set Identification (BSSID)* a um. Este método permite ao nó comunicar com todos os nós ao alcance do seu rádio, independentemente destes estarem associados a um WAVE *BSS* ou não[9]. Este método é denominado de *Outside the Context of a BSS* (OCB).

3.5 IEEE 1609.4

A secção 3.3 deste relatório referencia algumas limitações da camada física do 802.11p no que toca a interferências entre canais. Foi, então, necessário desenvolver métodos de atenuação desses problemas e é nesse contexto que apareceu o standard IEEE 1609.4.

O objetivo principal deste standard é o de fornecer operações entre diferentes canais, de forma a suportar diferentes tipos de aplicações, tais como aplicações de segurança e de entretenimento, por exemplo[15]. Esta coordenação entre canais torna-se possível através de uma interação entre a camada de LLC e a camada física, e providencia também suporte para o envio de *MAC Service Data Unit* (MSDU)[11].

O standard define, assim, quatro serviços [11]:

- *Channel routing* – Responsável pelo *routing* dos pacotes do LLC para o canal indicado, no contexto das operações de coordenação de canal definidas pela camada *MAC*
- Prioridade – A prioridade é usada para disputar o acesso ao meio, fazendo uso de *Enhanced Distributed Channel Access* (EDCA), adaptada a partir do protocolo 802.11e
- Coordenação de canal – Este serviço coordena os intervalos de sincronização dos canais, de acordo com as operações definidas pela camada *MAC*, de forma a transmitir as frames na frequência correta.
- MSDU *transfer service* – Este serviço consiste, na realidade, de três serviços distintos que gerem as comunicações do canal de controlo, do canal de serviços e do canal de transferência de dados. O objetivo destes serviços é fornecer uma prioridade superior ao WSMP.

Em suma, este *standard* define um esquema de divisão de tempo, de forma a alternar entre os canais atribuídos e, desta forma, suportar diferentes aplicações concorrentes. No entanto, existem alguns problemas que este *standard* não é capaz de colmatar efetivamente, que incluem o uso ineficiente dos canais, a ausência da noção de distância a uma área de cobertura de serviço e colisões sincronizadas no começo de um intervalo de um dado canal[16].

O maior problema do *standard* IEEE 1609.4 é a utilização dos canais. Por exemplo, um dispositivo que tenha acesso ao *Control Channel* (CCH) tem a oportunidade de enviar e receber mensagens de segurança rodoviária. Como tal, é esperado que o veiculo aloque as mensagens de segurança rodoviária durante os intervalos de uso do CCH, de forma a acomodar as necessidades de outros veículos que estejam, ativamente, a trocar de canais no *Service*

Channel (SCH). Na prática isto traduz-se no dobro da densidade de comunicação no CCH, o que significa que o desempenho da transmissão de mensagens de segurança rodoviária é afetado negativamente.[16]

Outro grande problema é a possibilidade de existirem colisões sincronizadas no início de um intervalo para acesso a um dado canal, se vários dispositivos dispõem de *frames* acumuladas e prontas a enviar. Um exemplo preocupante é o de aplicações de segurança rodoviária, onde bastantes mensagens são finalizadas e colocadas à espera para serem transmitidas no próximo intervalo de acesso ao CCH[16].

3.6 IEEE 1609.3

Este *standard* define as funcionalidades associadas com as camadas LLC, rede e transporte do modelo OSI e intitula-se de *WAVE networking services*[11]. Os mesmos dividem-se em dois serviços diferentes, os serviços de *data-plane* e os serviços de *management-plane*.

3.6.1 Serviços *data-plane*

Na *data-plane*, a arquitetura WAVE suporta duas *stacks* protocolares, nomeadamente IPv6 e WSMP, fornecendo assim suporte para aplicações de alta prioridade, sobre WSMP, e para aplicações tradicionais, sobre TCP/UDP/IP [11]. Desta forma, a *data-plane* é também responsável por lidar com a transmissão de *frames* IPv6 e WSMP, bem como a sua alocação para transmissão no canal correto e por gerir a prioridade das mensagens[14].

3.6.2 Wave Short Message Protocol

O WSMP é um protocolo desenvolvido a pensar nos requisitos exigentes das comunicações de segurança rodoviária, permitindo o envio de mensagens curtas, controlando diretamente os parâmetros do rádio, tais como o número do canal ou a energia de transmissão[17], [18], com vista a maximizar a probabilidade de receção atempada por todos os destinatários[11], [18].

O WSMP pode ser transmitido no CCH e no SCH, o que significa que pode ser transmitido sem que o veículo se encontre associado a um WBSS, funcionando assim em modo OCB. Neste modo, a aplicação que envia o pacote WSMP é registada na *Wave Management Entity* (WME) e endereça o pacote para um endereço MAC de *broadcast*. A camada MAC extrai do cabeçalho do pacote o número do canal, de forma a poder preparar a transmissão. Quando estes pacotes são recebidos, a *stack* protocolar do recetor faz chegar os dados à aplicação correspondente, baseando-se para isso no *Provider Service Identifier* (PSID) que se encontra no cabeçalho WSMP[17]. As implementações do WSMP devem, também, suportar *unicast*[19] e o reenvio de mensagens[11]. A Figura 4 ilustra o cabeçalho de uma mensagem WSMP.

Figura 4 - Cabeçalho de uma mensagem WSMP

3.6.3 Serviços management-plane

Os serviços de *management-plane* descritos no *standard* 1609.3 são responsáveis por configurar e manter o sistema. Isto implica, por exemplo, a configuração de IPv6, monotorização da utilização dos canais, a gestão de WBSS, e o registo de aplicações. Coletivamente, são conhecidos como *WAVE Management Entity* (WME)[11], [13].

Em particular, destacam-se o registo de aplicações, com o respetivo *provider service identifier* (PSID), e a gestão do WBSS. Desta forma, o WME é responsável por iniciar um WBSS para qualquer aplicação que forneça um serviço e, para tal, pode ser necessário efetuar trocas de chaves de encriptação, efetuar manutenção do estado de cada aplicação no contexto de um dado WBSS[11].

3.7 Outros componentes da arquitetura WAVE

Para além dos componentes descritos anteriormente, a arquitetura WAVE possui um *resource manager* (RM), definido pelo *standard* IEEE 1609.1, que permite a comunicação entre as aplicações que funcionam nas RSUs e nas OBUs. Existe, também, uma camada que fornece funcionalidades de segurança, definida no *standard* IEEE 1609.2, responsável por definir o formato, processamento e troca de mensagens seguras[13]. O *standard* IEEE 1609.5, por sua vez, é responsável por definir serviços de gestão de comunicações, suportando assim a conectividade entre as OBUs e as RSUs[12].

4 Long Term Evolution (LTE)

Nesta secção é descrito em detalhe a tecnologia LTE, abordando as suas características inerentes e arquitetura.

4.1 Introdução

O LTE é o último avanço na tecnologia de comunicação UMTS (*Universal Mobile Telecommunications System*), desenhado a pensar nas necessidades atuais dos utilizadores de redes móveis. Como tal, foram definidos, pela 3GPP (*Third Generation Partnership Project*) alguns objetivos fulcrais para este novo protocolo, tais como[20]:

- Aumento da capacidade de transferência de dados
- Custo, monetário, por bit mais reduzido, aumentando a eficiência do espectro utilizado
- Arquitetura de rede simplificada
- Mobilidade sem intervenção do utilizador, mesmo entre tecnologias diferentes
- Consumo energético razoável

O LTE foi pensado, desde o início, para suportar apenas serviços *Packet-Switched* (PS), em contraste com o seu antecessor, o *Global System for Mobile communications* (GSM), que

suportava apenas o modelo de serviços *Circuit-Switched* (CS). Desta forma, o LTE é capaz de fornecer conectividade IP entre o *User Equipment* (UE) e a *Packet Data Network* (PDN), sem qualquer adaptação na ligação de dados [20], sendo também capaz de funcionar em conjunto com tecnologias GSM e UMTS[20], [21].

A terminologia LTE diz respeito à componente de rádio, contudo, foi também necessário desenvolver tecnologias como a *System Architecture Evolution* (SAE), que inclui o *Evolved Packet Core* (EPC), que lidam com os outros componentes da rede. Em conjunto, os termos LTE e SAE formam o *Evolved Packet System* (EPS)[20], [21]. Estes conceitos serão abordados em detalhe nas próximas secções deste relatório.

4.2 Arquitetura da rede

A arquitetura do LTE divide-se em três componentes essenciais, nomeadamente o *User equipment* (UE), a *Evolved Universal Terrestrial Access Network* (E-UTRAN) e o *Evolved Packet Core* (EPC). Cada um destes componentes tem uma arquitetura interna própria, que irá ser discutida individualmente.

Na Figura 5 é possível ver a arquitetura geral de uma rede LTE.

4.2.1 *User Equipment*

O *User Equipment* (UE) representa o terminal de um utilizador, sendo normalmente um telemóvel. No entanto, noutros casos, o UE deve ser considerado como duas componentes em separado, o *mobile termination* (MT), responsável por gerir todas as funcionalidades de comunicação, e o *terminal equipment* (TE), responsável por terminar as *streams* de dados[22]. Uma placa de LTE externa, por exemplo, é o *mobile termination*, e um computador a que a placa se encontra ligada é o *terminal equipment*.

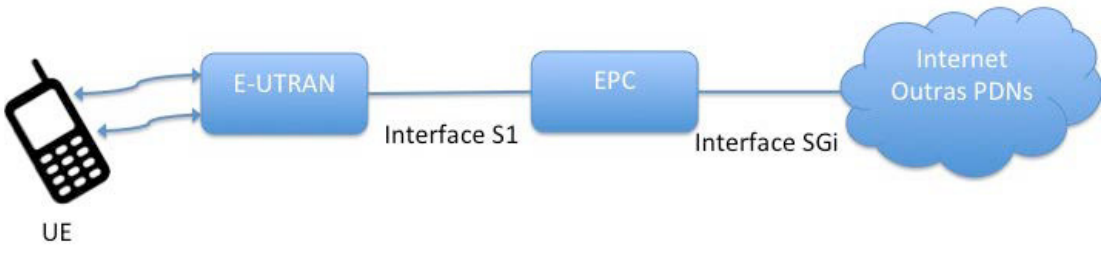


Figura 5 - Visão geral da arquitetura de uma rede LTE

4.2.2 Arquitetura da E-UTRAN

A E-UTRAN é responsável pelas comunicações rádio entre a componente móvel, o UE, e a EPC. A única componente da E-UTRAN é a *evolved Node B* (eNB), uma *base station* que controla os UE numa ou mais células[22].

A eNB é responsável pelo envio e receção de dados, descodificando os sinais analógicos e digitais da interface LTE. É também responsável pela gestão de operações de baixo nível, tal como o *handover* entre duas eNBs. Estas funcionalidades da eNB contribuem para a redução da latência associada com as trocas de informação entre a rede e o UE [22].

As eNBs estão ligadas ao EPC através de uma interface S1, podendo estar também ligadas com outras eNBs através de uma interface X2, usada, principalmente, para o reencaminhamento de pacotes durante o processo de *handover*[22].

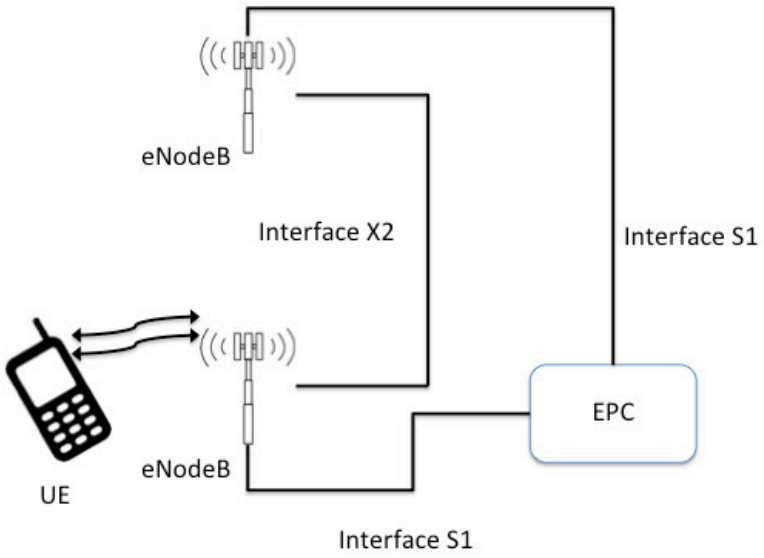


Figura 6 - Arquitetura da E-UTRAN

4.2.3 Arquitetura do EPC

A arquitetura do EPC envolve variados componentes, tais como o *Packet Data Network Gateway* (P-GW), o *Serving Gateway* (S-GW) e a *Mobility Management Entity* (MME) [20].

O P-GW é o ponto de contacto do EPC com o exterior. Através da interface SGi, cada PDN troca informação com um ou mais serviços externos ou PDN, tais como os servidores do operador da rede [22]. O P-GW é também responsável pela filtragem do trafego *downlink* do utilizador, atribuindo-lhe diferentes QoS *bearers* [20].

O S-GW funciona como um *router*, encaminhando os dados entre as eNBs e o *gateway* da PDN. Cada UE está associado a um S-GW, podendo mudar de S-GW se existir outro que, devido à

mobilidade do utilizador, esteja agora geograficamente mais próximo [22]. O S-GW serve, ainda, como para reter informação sobre os *bearers* quando o UE se move entre eNodeBs [20]. O S-GW realiza também algumas funções administrativas, tais como a recolha de informação relativa ao pagamento do serviço e é também capaz de interligar os UE com outras tecnologias 3GPP, tais como GPRS e UMTS [20].

O MME controla as operações de alto nível do UE, enviando-lhe mensagens não relacionadas com a comunicação do plano do utilizador, tais como a gestão de dados de uma *stream* ou sobre segurança. Uma rede contem, normalmente, vários MME geograficamente dispersos. Cada UE está associado a um dado MME, mas se o UE se afastar dessa zona, é-lhe atribuído outro MME.

O MME controla, ainda, outros elementos da rede, através da troca de mensagens internas com o EPC [22]. Estes protocolos são conhecidos como *Non-Access Stratum* (NAS), e são responsáveis por gerir os *bearers* e por gestão da comunicação [20].

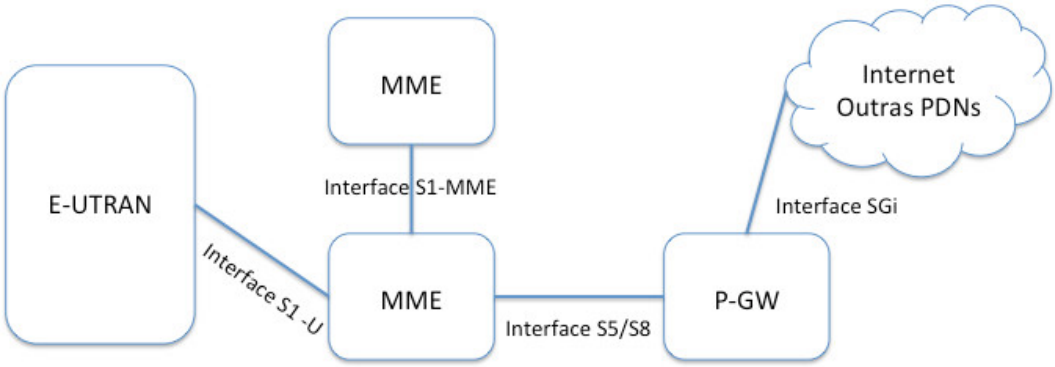


Figura 7 – Arquitetura simplificada do EPC

4.3 EPS *Bearer*

O LTE transporta os pacotes de dados através dos mesmos protocolos usados na Internet (IP,TCP,UDP), contudo necessita de mecanismos de transporte mais complexos do que os mecanismos normais. Necessita, assim, de lidar com dois problemas fulcrais: a mobilidade dos seus utilizadores e a qualidade de serviço [22].

De forma a colmatar estes problemas, o LTE transporta os dados de uma secção do sistema para outra recorrendo a EPS *bearers* [23], [24]. Um EPS *bearer* é um canal bidirecional que transfere dados na rota correta pela rede, com qualidade de serviço [22].

Existem dois tipos de *bearers*: os *default* e os *dedicated*. Quando um UE se liga a uma PDN é-lhe atribuído um *default bearer*, tal como ilustrado na Figura 8, fornecendo-lhe conectividade

com a rede. É então atribuído ao UE um endereço de IP, de forma a possibilitar a sua comunicação na rede [22].

Após obter o *default bearer*, um UE pode receber um ou mais *dedicated bearers*, que o ligam à mesma rede. Estes *dedicated bearers* não possuem um endereço de IP próprio e, como tal, partilham o endereço atribuído ao *default bearer*, adicionando ainda diferentes qualidades de serviço para os diferentes *bearers*. Um UE pode estabelecer um máximo de onze EPS *bearers*, o que lhe possibilita estabelecer várias ligações com diferentes políticas de qualidade de serviço [22].

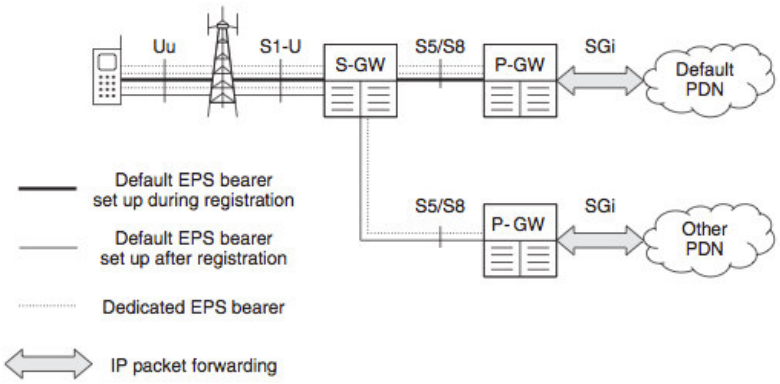


Figura 8 - Default e Dedicated EPS bearers[22]

4.4 Protocolos de comunicação

A camada protocolar do LTE tem dois planos, o plano do utilizador e o plano de controlo. O plano do utilizador gere os dados que são de interesse para o utilizador, enquanto que os protocolos no plano de controlo gerem a troca de mensagens entre os diferentes componentes da rede[22].

No que diz respeito ao plano do utilizador, quando um pacote IP destinado a um UE é recebido, este é encapsulado num protocolo especifico ao EPC e é enviado através de um túnel existente entre o P-GW e a eNodeB, para que esta o transmita para o UE [20], [22]. Para tal, é usado o protocolo *GPRS Tunneling Protocol* (GTP), nas interfaces S1 e S5/S8 [20]. A camada protocolar do plano do utilizador contém, ainda, as subcamadas *Packet Data Convergence Protocol* (PDCP), o *Radio Link Control* (RLC) e o *Medium Access Control* (MAC)[20], [21].

Relativamente ao plano de controlo, este é responsável por gerir a seleção da *Public Land Mobile Network* (PLMN), autenticação e gestão dos EPS *bearers*, entre outros.

5 Network Simulator 3 (ns-3)

Nesta secção será apresentado, de forma genérica, o simulador de redes ns-3. Este simulador foi uma parte central deste trabalho, pois permitiu o desenvolvimento da aplicação CarCoDe, necessária para o projeto, bem como o desenvolvimento do módulo de mobilidade. Este capítulo fará uma breve descrição do simulador, bem como da sua arquitetura. Também serão explicados os passos fundamentais para configuração de uma simulação no ns-3.

5.1 Descrição genérica

O *Network Simulator*, atualmente na sua terceira versão, o ns-3, é um simulador de redes bastante utilizado no âmbito de investigação. É um projeto *open source* e gratuito que pode ser executado nos sistemas operativos *Linux*, *Mac OS X* e *Windows* (através de *Cygwin*), desenvolvido essencialmente em C++ mas também com possibilidade de executar *scripts* em *Python*. A sua arquitetura é modular, tornando mais fácil e rápida a implementação de novos protocolos ou alteração de classes já existentes. Os investigadores partilham frequentemente o código desenvolvido, de forma a utilizar e extender o simulador, ajudando na adição de novos módulos, correção de erros e mostrando os resultados das suas experiências. Sendo a contribuição o espírito inerente à utilização do simulador, este é, portanto, fulcral na evolução do mesmo.

O ns-3 tem como grandes vantagens permitir o estudo de muitas tecnologias existentes de uma forma rápida, eficaz, com um alto nível de realismo e a um baixo custo. Desta forma, a comunidade associada não necessita de investir em tecnologias reais para realizar os seus estudos e experiências.

O ns-3 utiliza um modelo de simulação baseado em eventos discretos, isto é, cada evento (por exemplo, o envio de um pacote) é executado num dado instante temporal, em particular, marcando uma mudança no sistema. Entre eventos, assume-se que o sistema se mantém inalterado, pelo que este modelo permite ao simulador passar, temporalmente, de um evento para o próximo. Para além disso, possui mecanismos de *tracing* para obter facilmente mensagens de registo sobre o estado das simulações efetuadas. Também permite visualizar animações das simulações através de *NetAnim* e interação com sistemas reais, através de um sistema intitulado *Direct Code Execution* (DCE).

Por fim, torna-se relevante mencionar a excelente documentação que é disponibilizada, usando *Doxygen*, as APIs do simulador, sendo apresentadas de forma detalhada descrições sobre os métodos e atributos de cada classe do ns-3.

5.2 Arquitetura do ns-3

O ns-3 possuiu uma arquitetura modular, o que permite a um utilizador facilmente adicionar módulos que podem ser desenvolvidos de maneira independente do restante código. A arquitetura orientada a objetos, bem como os diferentes padrões e técnicas de programação utilizados contribuem para a robustez e facilidade de manutenção de todo o código. O ns-3 providencia, também, métodos de gestão de memória, através de uma implementação própria de *smart pointers* [25].

A Figura 9 reúne os elementos dos principais módulos do ns-3, ilustrando a forma como estes se encontram organizados. Os módulos *core* e *network* implementam os componentes mais genéricos, tais como *Event Scheduling* e *Callbacks* [25], enquanto que o modulo *network* implementa as abstrações relacionadas com a simulação de uma rede, fazendo destes dois módulos uma pedra basilar para a configuração de qualquer simulação. Os módulos que se encontram acima destes no esquema apresentado implementam os componentes específicos da rede da simulação. O módulo *internet*, por exemplo, implementa os protocolos ARP, IPv4, IPv6, TCP e UDP, enquanto que no módulo *Applications* podemos encontrar aplicações capazes de gerar tráfego, tais como *UdpClient, UdpServer, UdpEchoClient, UdpEchoServer, OnOffApplication* ou *PacketSink*. O ns-3 disponibiliza também módulos que permitem atribuir

mobilidade a um dado nó, que será abordado em mais detalhe numa próxima secção deste relatório, uma vez que foi fundamental ao desenvolvimento de uma parte deste projeto.

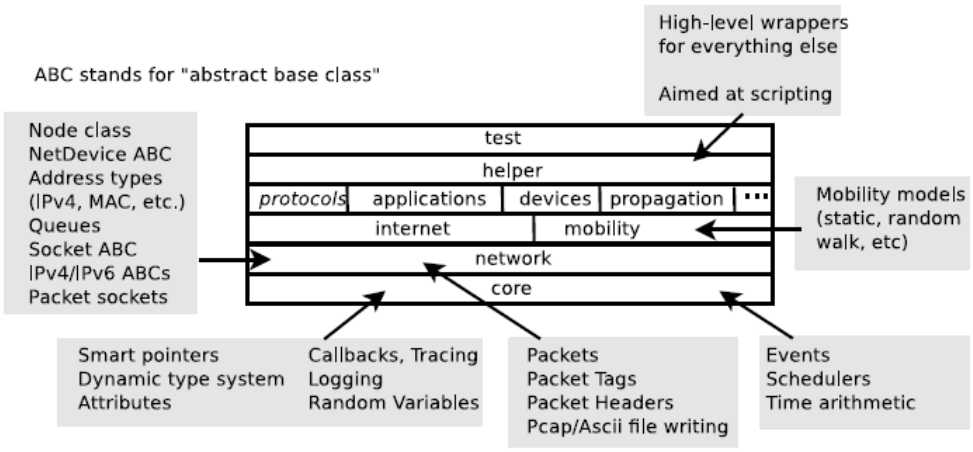


Figura 9 - Arquitetura modular do ns-3[26]

Alguns elementos são considerados fundamentais numa simulação do ns-3. As classes *Node*, *Packet*, *Socket,* Application, *NetDevices* e *Channels* implementam as abstrações fundamentais para que se possa simular o funcionamento de uma rede. Estas classes serão introduzidas brevemente nesta secção, sendo abordadas em mais detalhe numa próxima secção deste relatório.

A classe *Node* é uma classe base do ns-3, representativa de um nó numa rede, que pode ser instanciada, contendo apenas alguns atributos, tais como um número identificativo, uma lista de *NetDevices* e uma lista de *Applications*. A Figura 10 mostra, simplificadamente, a arquitetura da classe *Node* [25].

Esta arquitetura evita, assim, que exista um elevado número de dependências nas classes *Node, Application* ou *NetDevice*.

A classe *NetDevice* representa uma interface física num nó (uma interface de Ethernet, por exemplo), inspirado pela arquitetura de dispositivos do Linux. Uma implementação da classe *NetDevice* é forçada a implementar um apontador para um *Node*, um MacAddress, um MTU, uma *string* e ainda um booleano que determina se a interface está ligada ou desligada. Existem duas *callbacks*, que permitem o registo de uma função para o envio do pacote para as camadas protocolares superiores e o envio de uma notificação no caso de a placa alterar o seu estado, entre ligado e desligado[25]. O mecanismo de *callbacks* será explicado em mais detalhe na secção 5.3 deste relatório.

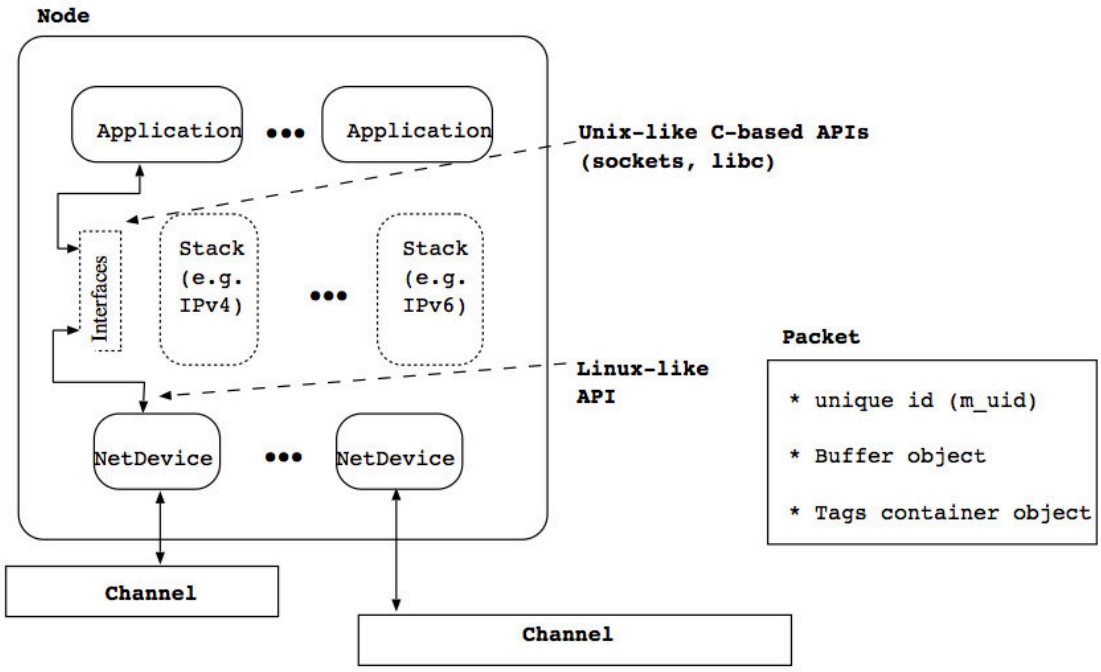


Figura 10 - Arquitetura da classe Node [25]

Um objeto do tipo *Packet* contêm um *buffer* de *bytes*. Este *buffer* contêm os cabeçalhos dos protocolos, através de rotinas de serialização e de remoção de serialização especificadas pelos utilizadores. Este *design* facilita a implementação de fragmentação de pacotes [25].

Um objeto do tipo *Application* é um conjunto de instruções que gera trafego para ser enviado pela rede simulada no ns-3. Estes objetos fazem uso de *sockets* para enviarem a informação pela rede.

5.3 Análise das classes do ns-3

Nesta secção são mostradas as classes do ns-3 relevantes para o trabalho proposto com uma detalhada análise às suas funções e atributos.

A análise do ns-3 revelou-se ser uma parte fundamental para o sucesso deste projeto. O simulador envolve um grande número de classes e, por isso, esta fase do trabalho foi particularmente exigente devido ao meu completo desconhecimento relativamente à estrutura do ns-3. Para isso, inicialmente foi seguido o tutorial disponibilizado pela equipa do ns-3, que apresenta alguns exemplos básicos, de forma a obter algumas noções de como configurar e executar uma simulação. Posteriormente, foram identificadas as potenciais classes existentes que poderiam ser utilizadas no trabalho a realizar e aprofundar os conhecimentos relativamente às funcionalidades, métodos e atributos inerentes às mesmas.

A Figura 11 apresenta um diagrama de classes simplificado, não detalhando os atributos e métodos de cada classe, para apenas demonstrar a relação entre as várias classes.

Seguidamente, é apresentada individualmente cada classe detalhando os seus métodos e atributos[25], [27] e feita uma breve descrição das mesmas.

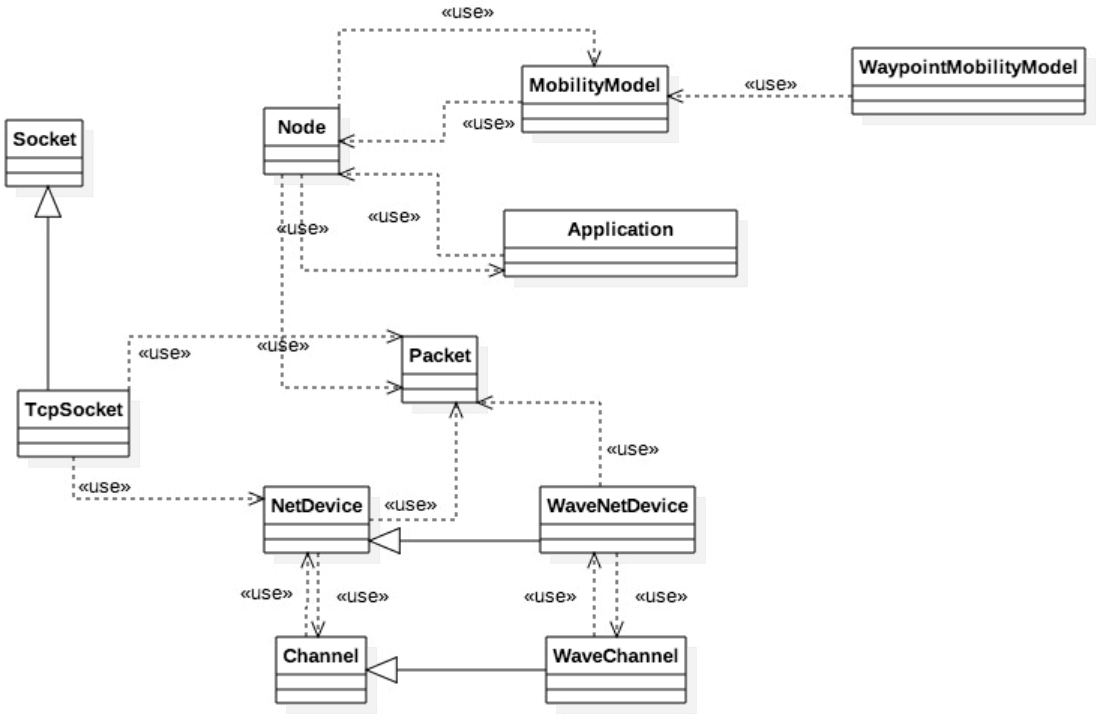


Figura 11 - Diagrama de classes simplificado das classes do ns-3 relevantes para o projeto

Node:

Um *Node* representa um nó na rede. Neste objeto existe uma lista de aplicações, onde são guardados os objetos, que representam as aplicações instaladas no nó, que interagem com o mesmo por meio da implementação de um `Socket`. Também existe uma lista onde são guardadas as interfaces de rede (`NetDevice`) instaladas no nó. Além disso, cada nó tem associado um ID único e também um ID de sistema utilizado para simulações de paralelismo.

Dois dos métodos mais utilizados e importantes nesta classe são os seguintes:

- `AddApplication` – Através desta função, é possível instalar uma aplicação no nó.
- `AddDevice` – Esta função permite associar um dispositivo de rede ao nó.

`Application`:

É a classe base para todas as aplicações do ns-3. Qualquer aplicação desenvolvida no simulador deve ser implementada a partir da classe `Application`. O utilizador deve definir nesta classe os atributos `StartTime` e `StopTime` que especificam o instante em que a aplicação inicia e o instante em que termina, respetivamente. Estes atributos são utilizados pelos seguintes métodos:

- `StartApplication` – Este método é chamado no instante inicial, definido no atributo `StartTime`, da execução de uma aplicação e tem de ser reescrito nas classes que implementam esta classe.
- `StopApplication` – Este método é chamado no instante definido pelo atributo `StoptTime`, assinalando o final da execução de uma aplicação. Também tem de ser reescrito nas classes que implementam esta classe.

`Socket`:

Esta classe é uma interface, baseada na API das *Berkeley sockets* (BSD), para protocolos de transporte, camada 4 do modelo OSI, como por exemplo TCP ou UDP, e também como interface para *packet sockets*. As funções mais utilizadas nesta classe são as seguintes:

- `Bind` – Esta função associa um endereço ao *socket*.
- `Connect` – Através desta função é iniciada a ligação do *socket* ao endereço de destino.
- `Recv` – Esta função permite a leitura de dados recebidos pelo *socket*.
- `Send` – Esta função é chamada quando se pretende enviar dados (`Packet`) para uma máquina remota.

No ns-3 existem implementações de vários tipos de s*ockets*, que herdam os atributos e métodos desta classe. Em seguida são enumeradas algumas dessas implementações:

- `TcpSocket` – Implementa um *socket* para comunicações que utilizem o protocolo TCP.
- `UdpSocket` – Implementa um *socket* em comunicações que utilizem o protocolo UDP.

- `PacketSocket` – Este tipo de *socket* permite uma ligação de dados entre uma aplicação e um dispositivo de rede instalado no nó.

`NetDevice`:

A classe `NetDevice` é uma interface que descreve a forma como a camada três do modelo OSI interage com a camada dois do mesmo modelo. Desta forma, todas as implementações da classe `NetDevice` emulam um dispositivo de rede real.

A classe `NetDevice` define, então, algumas funções importantes, tais como:

- `Send` – Esta função é responsável pelo envio de pacotes (*frames*) pelo `NetDevice`. Recebe como argumentos um `Packet` a enviar, um endereço MAC de destino e um inteiro que define o protocolo. Este último parâmetro facilita, mais tarde, a chamada do protocolo de camada 3 correto, na altura da receção do pacote.
- `SetReceiveCallback` – Esta função é responsável por associar uma `ReceiveCallback` a um dado `NetDevice`. Na prática, isto significa que, de cada vez que um pacote atinge este `NetDevice`, a `ReceiveCallback` passada por parâmetro é executada.

Alguns exemplos de `NetDevices` relevantes para este projeto são:

- `CsmaNetDevice` – Implementa uma interface de rede para ligações que utilizem o protocolo CSMA, usado em redes como *Ethernet* (IEEE 802.3).
- `WifiNetDevice` – Este dispositivo emula uma interface IEEE 802.11. Atualmente, o ns-3 suporta a camada física de 802.11a e 802.11b, variados modelos de propagação de sinal e modelos de atraso de propagação de sinal, o que confere algum realismo à modelação de redes que fazem uso deste `NetDevice`.
- `WaveNetDevice` – O modelo de WAVE foca-se na camada MAC, pelo que faz uso da camada física do 802.11a, preexistente. Neste contexto, é capaz de realizar comunicações baseadas no *standard* IEEE 802.11p, sendo capaz de transmitir em modo "*Outside the context of a Basic Service Set*"(OCB), uma funcionalidade central do *standard* IEEE 802.11p, que permite aos nós comunicarem sem estarem associados a um *Basic Service Set*. O `WaveNetDevice` será discutido, bem como o módulo de 802.11p em geral, na secção 5.5 deste relatório.

`Channel`*:*

É superclasse de um canal de rede, que faz a ligação entre duas interfaces de nós distintos. A sua implementação está diretamente relacionada com o tipo de rede. Deste modo deve ser implementada com o mesmo tipo do `NetDevice`.

Alguns exemplos de `Channel` são:

- `CsmaChannel` – Implementa a ligação por cabo entre dois nós numa rede CSMA (*Ethernet*).
- `WifiChannel` – Esta classe implementa as propriedades físicas de um canal *Wifi* real. É usada em conjunto com a classe `WifiPhy`.

`Packet`:

Os pacotes modelados contêm um *buffer* de *bytes* e um conjunto de *byte tags* e *packet tags*. O *buffer* de bytes contem a informação serializada do pacote (por exemplo, *headers*, *trailers* e dados), o que permite facilmente modelar um dado protocolo de interesse para o utilizador. É esperado que a estrutura do pacote seja implementada de forma a corresponder, *bit* a *bit*, ao pacote real, contudo, os utilizadores podem escolher modelar pacotes como desejarem, de forma a irem ao encontro do objetivo das suas investigações.

`Callback`:

As *callbacks*, no ns-3, fazem uso de uma funcionalidade da linguagem C chamada *pointer to function returning integer*. Este método permite ao ns-3 obter um acoplamento mais baixo, uma vez que deixa de ser necessário instanciar objetos sem relação.

Esta prática faz com que seja possível associar, em tempo de execução, um conjunto de instruções a executar quando um dado evento é ativado. Esta funcionalidade é usada, por exemplo, no tratamento dos pacotes que chegam a um dado `NetDevice`. O utilizador pode, então, especificar a forma como quer tratar os pacotes, conferindo uma grande flexibilidade ao simulador.

`Simulator`:

Este é das classes mais importantes do ns-3. Como já referido no capítulo 5.1, o ns-3 é baseado em eventos discretos, ou seja, o simulador mantem uma fila de eventos que vão sendo executados numa ordem sequencial de tempo. A classe Simulator é responsável por manipular todos os eventos na simulação. Em seguida são apresentadas as principais funções desta classe:

- `Run` – Executa a simulação definida. O simulador vai executar todos os eventos escalonados na simulação até que estes se esgotem ou que seja chamada a função `Stop` pelo utilizador.
- `Stop` – Quando esta função é chamada, mais nenhum evento é executado na simulação.
- `Schedule` – Esta função permite ao utilizador escalonar um evento para um determinado tempo relativo na simulação. Quando a simulação chega ao tempo relativo especificado, é chamada a função associada ao evento escalonado.
- `Cancel` – Permite cancelar um determinado evento escalonado.
- `Now` – Ao chamar esta função é possível obter o instante temporal relativo da simulação.

***Tracing*:**

O ns-3 oferece aos seus utilizadores variadas formas de obter informação a partir das suas simulações. A maioria dos seus módulos possui métodos de *tracing* implementados, tais como ficheiros PCAP, que podem ser usados, em conjunto com programas como o *Wireshark* ou o *Tcpdump*, para analisar os pacotes recebidos e transmitidos por uma dada interface.

De forma a poder acomodar as necessidades dos utilizadores, é possível definir funções a serem executadas quando um determinado evento ocorre (por exemplo, quando um pacote está pronto para ser transmitido pela camada física). O utilizador pode, assim, registar uma variedade de eventos que ocorrem durante uma simulação, bem como as propriedades desse evento (por exemplo, a informação de um pacote que é aceite pela camada física mas rejeitado pela camada MAC)

O ns-3 possui, também, um modelo de *logging*, que é um requerimento de qualquer módulo implementado. Como tal, todos os módulos possuem algum tipo de *logging*, o que facilita a

depuração de uma simulação, bem como a análise dos eventos à medida que estes são processados pelo ambiente de simulação. Este método imprime informação para o descritor de ficheiros *stderr* e possui sete níveis de severidade, de forma a não sobrecarregar o utilizador com informação.

Modelo de Erros:

De forma a obter um maior realismo, a maioria das tecnologias implementadas possuem algum tipo de modelo de erros, de forma a modelar os problemas que poderão surgir durante a comunicação. Estes modelos podem ir desde modelos simples, que calculam a possibilidade de erros aleatoriamente, a modelos de erros complexos e específicos para uma dada tecnologia que, pela sua elevada complexidade computacional, são, por vezes, computados previamente e distribuídos com o simulador.

5.4 Criação de simulações no ns-3

As simulações no ambiente do ns-3 são executadas a partir de um *script* de simulação, que define quais são os elementos necessários à realização da mesma. É neste *script* de simulação que o utilizador especifica a topologia de rede, as aplicações dos diferentes nós, os mecanismos de *tracing* e a mobilidade dos nós, entre outros.

Existem, portanto, alguns passos importantes, a seguir enumerados:

- Incluir as bibliotecas necessárias para a simulação
- Ativar mensagens de *logging*, caso pretendido
- Criar os nós da rede
- Criar os *NetDevices* em cada nó
- Associar um *Channel* a cada *NetDevice*
- Instalar uma pilha protocolar em cada nó
- Atribuir endereços às interfaces de rede
- Instalar aplicações em cada nó e definir os tempos de início e fim das mesmas
- Ativar mecanismos de *tracing*
- Executar a simulação

Durante a componente de análise ao simulador, deste projeto, foi estudado o tutorial [28] disponibilizado pelo consórcio ns-3. Este detalha alguns exemplos com vista a esclarecer o utilizador no uso das ferramentas do mesmo. De forma a ilustrar a criação de uma simulação simples, é apresentado na Figura 12 o código do primeiro exemplo, *first.cc*, que define uma simulação onde é trocado tráfego entre dois nós com ligação ponto-a-ponto.

```
#include "ns3/core-module.h"
#include "ns3/network-module.h"
#include "ns3/internet-module.h"
#include "ns3/point-to-point-module.h"
#include "ns3/applications-module.h"

using namespace ns3;

NS_LOG_COMPONENT_DEFINE ("FirstScriptExample");

int
main (int argc, char *argv[])
{
  Time::SetResolution (Time::NS);
  LogComponentEnable ("UdpEchoClientApplication", LOG_LEVEL_INFO);
  LogComponentEnable ("UdpEchoServerApplication", LOG_LEVEL_INFO);

  NodeContainer nodes;
  nodes.Create (2);

  PointToPointHelper pointToPoint;
  pointToPoint.SetDeviceAttribute ("DataRate", StringValue ("5Mbps"));
  pointToPoint.SetChannelAttribute ("Delay", StringValue ("2ms"));

  NetDeviceContainer devices;
  devices = pointToPoint.Install (nodes);

  InternetStackHelper stack;
  stack.Install (nodes);

  Ipv4AddressHelper address;
  address.SetBase ("10.1.1.0", "255.255.255.0");

  Ipv4InterfaceContainer interfaces = address.Assign (devices);

  UdpEchoServerHelper echoServer (9);

  ApplicationContainer serverApps = echoServer.Install (nodes.Get (1));
  serverApps.Start (Seconds (1.0));
  serverApps.Stop (Seconds (10.0));

  UdpEchoClientHelper echoClient (interfaces.GetAddress (1), 9);
  echoClient.SetAttribute ("MaxPackets", UintegerValue (1));
  echoClient.SetAttribute ("Interval", TimeValue (Seconds (1.0)));
  echoClient.SetAttribute ("PacketSize", UintegerValue (1024));

  ApplicationContainer clientApps = echoClient.Install (nodes.Get (0));
  clientApps.Start (Seconds (2.0));
  clientApps.Stop (Seconds (10.0));
  Simulator::Run ();
  Simulator::Destroy ();
  return 0;
}
```

Figura 12 - Código do exemplo first.cc do tutorial do ns-3

Este exemplo inclui os passos principais de um *script* de simulação. Inicialmente são incluídos os cabeçalhos necessários a esta simulação e, de seguida, é declarado o *namespace ns3*, uma vez que todo o código implementado no ns-3 se encontra neste *namespace*.

Seguidamente, são ativados os componentes de logging, mecanismos fundamentais para recolher informação sobre o estado da simulação. A instrução `NS_LOG_COMPONENT_DEFINE` permite definir um componente de *logging*, neste caso, o *FirstScriptExample*. Existem, também, instruções dentro da função *main* que ativam o componente de *logging*, nomeadamente para as aplicações *UdpEchoClientApplication* e *UdpEchoServerApplication*:

```
LogComponentEnable("UdpEchoClientApplication",LOG_LEVEL_INFO);
LogComponentEnable("UdpEchoServerApplication",LOG_LEVEL_INFO);
```

De seguida, inicia-se a construção da rede propriamente dita, criando os nós. O ns-3 disponibiliza um conjunto de contentores, entre eles o `NodeContainer`, que auxiliam na criação e manipulação de certos objetos. Através da função `Create`, são criados dois nós, que são então armazenados na lista do objeto do tipo `NodeContainer`. Após a criação dos nós da rede, o passo seguinte foca-se na definição da topologia da rede.

Os *helpers* são assistentes que facilitam este processo e, para definir a ligação ponto-a-ponto é usado o `PointToPointHelper`. Este assistente é capaz de configurar atributos dos dispositivos e canais ponto-a-ponto, através dos métodos `SetDeviceAttribute` e `SetChannelAttribute`. O método `Install` recebe como parâmetro os nós guardados no `NodeContainer` e instala em cada um deles uma interface de rede ponto-a-ponto sendo estas, por sua vez, guardadas num contentor próprio, o `NetDeviceContainer`.

As instruções seguintes dizem respeito à instalação da pilha protocolar e a atribuição de endereços às interfaces dos nós. Para isso, recorre-se novamente ao auxílio dos *helpers* `InternetStackHelper` e `Ipv4AddressHelper`, respetivamente através dos métodos `Install` e `Assign`. Por fim, resta apenas criar e instalar as aplicações em ambos os nós. Num dos nós é instalado uma aplicação `UdpEchoServer` e noutro uma aplicação `UdpEchoClient` através do método `Install` de cada um dos *helpers* respetivos, `UdpEchoServerHelper` e `UdpEchoClientHelper`. Pelo método `SetAttribute` é possível configurar alguns atributos das aplicações, como é visível no código do exemplo. Também é necessário indicar o tempo do início e do fim das aplicações, através dos métodos `Start` e `Stop`. Para executar a simulação falta apenas acrescentar a instrução `Simulator::Run ()` que executará todos os eventos escalonados pelas aplicações nos

tempos relativos da simulação, definidos previamente. Após isso, torna-se necessário destruir os objetos que foram criados para a simulação através da instrução `Simulator::Destroy ()`.

5.5 Módulo de 802.11p

O módulo de 802.11p implementado no ns-3 merece ser destacado neste relatório, uma vez que foi sobre este módulo que uma parte significativa do trabalho descrito neste relatório foi efetuada.

Durante a fase de análise do simulador, foi descoberto que, na versão do simulador usada no desenvolvimento deste trabalho (versão 3.19), o nível de maturidade do modelo de WAVE era ainda baixa, com muitas das funcionalidades essenciais ao desenvolvimento do projeto em falta. Este facto levou ao uso de uma versão experimental do módulo, que acabou por ser incluída na distribuição oficial do ns-3 na versão 3.21.

O módulo de WAVE foca-se, essencialmente, na camada MAC e na camada de coordenação entre canais[29] (IEEE 1609.4, descrito na secção 3.4 deste relatório). A decisão de focar o desenvolvimento na camada MAC deve-se ao facto de ser necessário comunicar sem estar associado a um WBSS, ou seja, comunicar em modo OCB, para que a implementação esteja de acordo com o especificado no *standard*. Uma vez que a camada física do 802.11p é muito semelhante à camada física do 802.11a, o módulo de WAVE usa a camada física do módulo de wifi implementado no simulador [29]. A classe `Wifi80211pHelper` permite ao utilizador configurar a camada física para que esta funcione a 10MHz ou a 20MHz, o que, na prática, simula o funcionamento real de um rádio 802.11p.

Este módulo implementa as funcionalidades de coordenação de canal, através da classe `ChannelCoordinator`, e *channel routing*, entre outras, o que permite ao módulo simular as transições entre os SCH e o CCH, bem como os Guard Intervals. Estas transições assumem a sincronização temporal perfeita para todos os nós da rede.

5.5.1 Limitações

Tal como referido anteriormente, este módulo sofre, ainda, de muitos problemas de maturidade. Na sua versão atual ainda não são modelados os outros aspetos dos *standards* IEEE 802.11p e IEEE 1609.X não mencionados nesta secção[29], tais como o IEEE 1609.1, IEE 1609.2.

Este módulo também não considera os aspetos de mobilidade veicular e, embora seja possível adicionar qualquer modelo de mobilidade implementado no ns-3, não suporta modelos de propagação e de erros específicos a estas redes [29].

5.6 Módulo de LTE

O módulo de LTE, em contraste com o módulo de 802.11p, é um módulo bastante maduro e ativamente mantido. O módulo suporta uma grande maioria das funcionalidades do LTE especificado pela 3GPP, bem como modelos de propagação e de perda de sinal específicos [30]. O módulo foi criado a pensar na avaliação de aspetos específicos do LTE, tais como a gestão dos recursos de rádio e alocação de pacotes com qualidade de serviço, entre outros. A Figura 13 mostra uma visão geral da arquitetura do módulo de LTE implementado no ns-3.

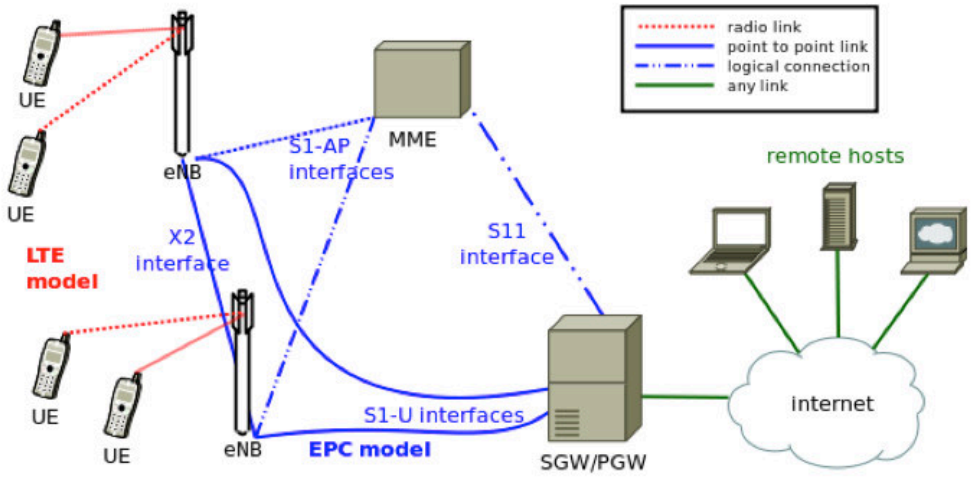


Figura 13 - Visão geral da arquitetura do módulo de LTE implementado[30]

O módulo suporta o uso de dezenas de eNBs e centenas de UEs numa única simulação, algo que só é possível pelo uso de uma granularidade ao nível do *Resource Block* (RB), uma vez que o uso de um nível mais baixo, necessitaria de um poder computacional muito elevado [30].

Relativamente ao EPC modelado, uma das decisões tomadas durante o desenvolvimento deste módulo foi o de implementar o SGW e o PGW num único nó, uma vez que as comunicações entre SGW não são abordadas por este módulo. Na prática, isto significa que são removidas as interfaces S5 ou S8, especificadas pela 3GPP. O módulo foca-se na modelação do plano de dados, sendo a modelação do plano de controlo feita através de chamadas diretas aos objetos. Apenas um tipo de PDN é suportado, o IPv4 [30].

O modelo de atenuação de sinal incluído neste módulo é baseado em cálculos previamente efetuados em MATLAB, de variáveis como a velocidade dos utilizadores e o número de

resource blocks a considerar, entre outros. A equipa optou por incluir apenas três cenários distintos: urbano, veicular e pedestre, assumindo velocidades de 0 e 3 km/h; 0, 30 e 60 km/h; e 3 km/h, respetivamente [30], de forma a limitar a complexidade computacional do simulador e a diminuir o uso de memoria.

5.7 Módulos de mobilidade

A mobilidade dos nós é um fator essencial para as investigações de VANETs ou *Mobile Ad-hoc Networks* (MANETs) uma vez que estas redes se caracterizam, entre outros fatores, pela alta mobilidade dos seus nós. Neste contexto foram implementados modelos de mobilidade sintéticos no ns-3, tais como o modelo de mobilidade Gauss-Markov e o modelo de mobilidade *Random Waypoint*, que permitem a modelação de redes com nós móveis [31]. O ns-3 possui ainda um módulo que permite a importação de ficheiros de mobilidade gerados em formato ns-2, uma vez que a maioria dos simuladores de trânsito é capaz de exportar ficheiros de mobilidade neste formato, permitindo assim que seja gerada uma mobilidade complexa num simulador de trânsito, como por exemplo, o *Simulation of Urban MObility* (SUMO), e usar essa mobilidade com o ns-3.

No decorrer deste projeto foi, ainda, desenvolvido, ao abrigo de um programa de Verão patrocinado pelo *ns-3 consortium*, um modelo de mobilidade realístico, o RoutesMobilityModel [32], cujas funcionalidades e implementação serão descritas em mais detalhe na secção 7.1 deste relatório.

5.7.1 Modelos de mobilidade sintéticos

Os modelos de mobilidade sintéticos são modelos aleatórios, que tentam oferecer uma mobilidade aproximada da realidade [33]. Estes modelos, contudo, não tomam em consideração a rede de estradas e, como tal, não são adequados, em particular, para modelar redes veiculares [34]. Algumas investigações, em particular, relacionam o uso destes modelos sintéticos com um desempenho irreal dos protocolos de *routing* como o *Ad-hoc On Demand Distance Vector* (AODV) ou o *Optimized Link State Routing* (OLSR) [35], [36], [34], especialmente desenvolvidos para redes com nós móveis.

O ns-3 possui implementações de vários modelos de mobilidade [31], tais como:

- Gauss-Markov – O modelo de mobilidade Gauss-Markov atribui uma velocidade e direção a um dado nó, gerando a mobilidade através de uma atualização da velocidade e direção do nó a um intervalo de tempo especificado pelo utilizador. Este módulo permite que as direções e velocidades passadas influenciem diretamente a escolha

das futuras velocidades e direções [37], o que melhora significativamente o seu desempenho quando comparado com outros modelos de mobilidade existentes no ns-3.

- *Random Waypoint* – Este modelo de mobilidade escolhe a velocidade e o destino aleatoriamente, em determinadas partes da sua trajetória. Desta forma, os nós movem-se com velocidade constante entre estes dois pontos [38]. De cada vez que um nó chega a um destino, é escolhida novamente uma nova velocidade e um novo destino.

- *Random Walk 2D* – Este modelo de mobilidade escolhe uma velocidade e um destino aleatório, até que seja percorrida uma determinada distancia ou até que o tempo especificado expire. No caso de a trajetória de um dado nó encontrar uma das fronteiras especificadas pelo utilizador a trajetória do nó é refletida dessa barreira, o que, na pratica, significa que o nó se mantem dentro das coordenadas pretendidas pelo utilizador [39].

Estes modelos caracterizam-se, em particular, pela elevada facilidade com que podem ser configurados em qualquer tipo de simulação.

5.7.2 Simulação de mobilidade urbana: SUMO

O SUMO é um simulador de trânsito *open-source*, disponível desde 2001, capaz de modelar sistemas de trânsito intermodais, incluindo veículos, transportes públicos e pedestres. A *suite* de *software* SUMO inclui ainda ferramentas capazes de encontrar rotas, visualização da simulação, importação de redes e cálculo de emissões de gases[40]. O SUMO oferece ainda APIs capazes de controlar remotamente a simulação, tais como a API TraCI [41].

O SUMO é, assim, capaz de gerar uma mobilidade altamente realística, contudo, a sua configuração é bastante complexa [42] e o seu suporte e documentação são bastante pobres, o que dificulta o seu uso por parte de investigadores cujo foco é na camada de comunicação.

O SUMO é capaz de gerar um ficheiro de mobilidade em formato ns-2, que pode ser importado para o ns-3 através da classe `Ns2MobilityHelper`. Desta forma, é possível gerar mobilidade realística e usar essa mobilidade com o simulador de redes. É ainda possível fazer com que as comunicações dos nós alterem a mobilidade dos mesmos, como, por exemplo, para forçar alguns veículos a escolherem um caminho diferente no caso de existir informação na rede relativa a um acidente na rota desses veículos, através do uso da API TraCI.

Este simulador será usado para comparar o desempenho da comunicação de dados com mobilidade gerada pelo `RoutesMobilityModel`, desenvolvido no âmbito deste projeto, e com mobilidade gerada através do uso do SUMO em conjunto com o ns-3. Esta comparação será abordada em maior detalhe na secção 8.1 deste relatório.

5.7.3 WaypointMobilityModel

O `WaypointMobilityModel` fornece mobilidade baseada em *waypoints* e, como tal, contêm a totalidade de *waypoints* que caracterizam a sua mobilidade. Cada *waypoint* representa a localização, de um dado nó, a um dado momento e, uma vez que os nós se movem a uma velocidade constante entre os *waypoints*, cada objeto é capaz de determinar a sua posição e velocidade, por interpolação.

Uma vez que este modelo de mobilidade requer apenas a especificação de um conjunto de *waypoints*, existe uma classe, `Waypoint`, que representa os objetos a adicionar a este modelo de mobilidade. De forma a criar um objeto desta classe é, apenas, necessário especificar a posição e o tempo em que o nó deve atingir esse *waypoint*.

Este modelo de mobilidade, embora flexível e altamente configurável, é inadequado para a geração de mobilidade complexa, uma vez que o utilizador teria que especificar manualmente todos os *waypoints* de cada nó.

Neste contexto foi desenvolvido o `RoutesMobilityModel`, que faz uso deste modelo de mobilidade, em conjunto com serviços de informação externos, de forma a gerar rapidamente mobilidade para vários nós sem configurações morosas por parte do utilizador.

5.7.4 Contribuições de código para o ns-3

Sendo o ns-3 um projeto *open-source*, as contribuições de código, sejam novas funcionalidades, correções a problemas ou até adição de documentação, são encorajadas e apoiadas pela equipa e pelos utilizadores, em geral.

No contexto deste projeto foram submetidos dois *patches*, contendo a adição do módulo desenvolvido e uma subsequente revisão do mesmo, com novas funcionalidades, que será abordado em detalhe na secção 7.1 deste relatório. Foram, também, submetidas pequenas alterações à documentação, efetuadas durante um *sprint* de documentação organizado pela equipa do ns-3.

De forma a submeter código ao ns-3, este necessita de respeitar algumas convenções estabelecidas pela equipa [43]. É exigido do utilizador que desenvolva todo o código com base no estilo de codificação do ns-3. É, também, pedido ao utilizador que documente o novo

código com exemplos e documentação, e que desenvolva ou atualize os testes unitários, conforme apropriado.

De forma a manter um *software* robusto, este código é ainda avaliado antes da sua inclusão numa versão do ns-3. O código é, então, submetido à plataforma *rietveld* disponibilizada, que possibilita, à equipa e ao programador, efetuar correções, que contribuem para a consistência do *software* final, bem como para a sua robustez.

6 Serviços do Google Maps

As *Application Programming Interfaces* (APIs) do Google Maps são uma coleção de interfaces HTTP para acesso a serviços do Google Maps, capazes de oferecer dados geográficos para aplicações. Existe uma variada oferta de APIs que fornecessem diferentes informações, tais como a *Directions API, Places API, Roads API, Elevation API,* entre outras[44]. Neste capítulo iremos abordar a *Directions API* e a *Places API*, uma vez que estas se revelaram centrais para o desenvolvimento do RoutesMobilityModel, um modelo de mobilidade realístico, desenvolvido no âmbito deste projeto de estágio.

6.1 *Directions* API

A *Directions* API fornece o cálculo de uma rota entre dois lugares, através de um pedido HTTP. As rotas fornecidas podem ser calculadas para trajetos a pé, utilizando transportes públicos ou de carro [45].

Esta API requere dois parâmetros para o seu funcionamento, a origem e o destino, em adição à chave da API. O programador pode ainda passar alguns parâmetros opcionais, tais como o modo de transporte, se existe algum tipo de estrada que se pretende evitar e hora de partida, entre outros. A API responde com os pontos geográficos que compõem a rota selecionada, bem como diferentes informações relativamente ao trajeto.

A API existe em duas versões, a paga e a gratuita. Com a API gratuita, o utilizador pode pedir, apenas, 2500 pedidos de direções por cada 24 horas. O utilizador está ainda restringido a 2 pedidos por segundo, bem como a um total de oito pontos intermédios no cálculo da rota. A versão paga suporta 100 000 pedidos de direções por cada 24 horas, 23 pontos intermédios e 10 pedidos por segundo. Esta versão disponibiliza ainda informações sobre o estado do trânsito, se disponível na área selecionada.

A resposta da API contem um atributo de controlo, o atributo `status`, que possui diferentes valores consoante o estado da API ou da conta do utilizador, permitindo assim perceber se existe algum problema por parte da API que torne impossível satisfazer o pedido. Pode conter os valores `OK`, `NOT_FOUND`, `ZERO_RESULTS`, `MAX_WAYPOINTS_EXCEEDED`, `INVALID_REQUEST`, `OVER_QUERY_LIMIT`, `REQUEST_DENIED` e `UNKNOWN_ERROR`.

A Figura 14 ilustra uma resposta simplificada desta API, bem como o URL usado como pedido, que especifica os paramêtros necessários para o pedido.

```
https://maps.googleapis.com/maps/api/direc-
tions/xml?origin=Rua%20S.%20Tomé,%20Porto,%20Portugal&destination=Rua%20Dr.%20Rob-
erto%20Frias,%20Porto,%20Portugal&sensor=false&mode=driving&depar-
ture_time=1436751913&key=AIzaSyDbibkYMhqBnCzWaEJUgMsuySpoOmgNy5c

<?xml version="1.0" encoding="UTF-8"?>
<DirectionsResponse>
 <status>OK</status>
 <route>
  <summary>A42</summary>
  <leg>
   <step>
    <travel_mode>DRIVING</travel_mode>
    <start_location>
     <lat>41.3896460</lat>
     <lng>-8.2290946</lng>
    </start_location>
    <end_location>
     <lat>41.3895973</lat>
     <lng>-8.2291966</lng>
    </end_location>
    <polyline>
     <points>i|r{Fxffq@BBDP</points>
    </polyline>
    <duration>
     <value>1</value>
     <text>1 min</text>
    </duration> <html_instructions>Head&lt;b&gt;south-
west&lt;/b&gt;on&lt;b&gt;EM563&lt;/b&gt;toward &lt;b&gt;CM1160-
1&lt;/b&gt;</html_instructions>
    <distance>
     <value>10</value>
     <text>10 m</text>
    </distance>
   </step>
<step>(...)</step>
<step>(...)</step>
<step>(...)</step>
   <duration>
    <value>2514</value>
    <text>42 mins</text>
   </duration>
   <distance>
```

```
<value>57433</value>
    <text>57.4 km</text>
   </distance>
   <start_location>
    <lat>41.3896460</lat>
    <lng>-8.2290946</lng>
   </start_location>
   <end_location>
    <lat>41.1772191</lat>
    <lng>-8.5995929</lng>
   </end_location>
   <start_address>Rua, 4610, Portugal</start_address>
   <end_address>Rua Doutor Roberto Frias, 4200 Porto, Portugal</end_address>
  </leg>
  <copyrights>Map data ©2015 Google</copyrights>
  <overview_polyline>
   <points>i|r{Fxffq@z@}@`(...)@`BXL~B~@</points>
  </overview_polyline>
  <bounds>
   <southwest>
    <lat>41.1769620</lat>
    <lng>-8.5995929</lng>
   </southwest>
   <northeast>
    <lat>41.3896460</lat>
    <lng>-8.1932919</lng>
   </northeast>
  </bounds>
 </route>
</DirectionsResponse>
```

Figura 14 - Codigo exemplo de resposta da Google Maps Directions API

6.2 *Places* API

A *Places* API é um *web service* que retorna informação sobre locais de interesse, como locais de comércio e serviços, localizações geográficas e pontos de interesse, através de pedidos HTTP[46]. Esta API suporta uma variedade de pesquisas à base de dados de locais do Google, tal como obter uma lista com locais na proximidade de um dado ponto geográfico, obter detalhes sobre um local em particular, adicionar locais, entre outras.

A pesquisa de locais nas proximidades de um dado ponto geográfico permite ao utilizador obter uma lista com locais, num raio de 5 quilómetros, de um dado ponto. De forma a obter esta informação, a API requer uma chave, uma localização central para a pesquisa e o raio da pesquisa. Opcionalmente, o utilizador pode ainda fornecer parâmetros adicionais, que permitem um maior controlo sobre a informação obtida.

Esta API possui limitações ao seu uso, sendo apenas possível realizar 1000 pedidos por cada 24 horas. A API limita, também, a sua resposta a um total de 60 locais por pesquisa, divididos por páginas de 20 locais cada. A divisão dos resultados por páginas permite um maior controlo sobre a quantidade de informação retornada, sendo que cada nova página representa um novo pedido à API.

A resposta da API possui informações sobre variados locais, encontrados de acordo com o especificado pelo utilizador. Entre estas informações inclui-se, em particular, as coordenadas geográficas do local e um atributo especial, o `pagetoken`, que permite ao utilizador aceder aos próximos 20 locais. A resposta da API possui, ainda, um atributo `status`, semelhante ao descrito na secção 6.1. A Figura 15 ilustra uma resposta desta API.

```
https://maps.googleapis.com/maps/api/place/nearbysearch/xml?location=41.177891,-8.610778&radius=5000&key=AIzaSyDbibkYMhqBnCzWaEJUgMsuySpoOmgNy5c

<PlaceSearchResponse>
 <status>OK</status>
<result>
  <name>Coliseu do Porto</name>
  <vicinity>Rua de Passos Manuel 137, Porto</vicinity>
  <type>point_of_interest</type>
  <type>establishment</type>
  <geometry>
   <location>
    <lat>41.1469920</lat>
    <lng>-8.6054170</lng>
   </location>
  </geometry>
  <rating>4.1</rating>
  <icon>(...)</icon>
  <reference>(...)</reference>
  <id>(...)</id>
  <photo>(...)</photo>
  <place_id>ChIJY7GRoOVkJA0RPzOD00nkdXI</place_id>
  <scope>GOOGLE</scope>
 </result>
<result>(...)</result>
<result>(...)</result>
<next_page_token>CvQB7AAA(...)AaeZ3OvS8O_</next_page_token>
</PlaceSearchResponse>
```

Figura 15 - Exemplo simplificado de resposta da Places API

7 Descrição técnica

Neste capítulo é feita a descrição técnica do trabalho desenvolvido, nomeadamente, a modelação e implementação de um novo modelo de mobilidade no ns-3, o RoutesMobilityModel, bem como a implementação e modelação de uma aplicação de disseminação de dados, a aplicação CarCoDe. É também efetuada uma análise às classes existentes no ns-3 relevantes para o desenvolvimento do projeto.

7.1 RoutesMobilityModel

O RoutesMobilityModel é um modelo de mobilidade realístico, desenvolvido para o simulador ns-3 ao abrigo de um programa de Verão tutorado por Tom Henderson, atual líder do consórcio ns-3, e Scott Carpenter. O objetivo deste modelo de mobilidade é o de oferecer ao utilizador a capacidade de, facilmente, gerar mobilidade realística no ns-3, recorrendo, para isso, a serviços de planeamento de viagens externos.

7.1.1 Levantamento de requisitos

O levantamento de requisitos é um processo importante na fase de análise de um projeto, uma vez que permite definir claramente a abordagem a tomar no desenvolvimento da solução final. O trabalho descrito neste capítulo foi alvo de um levantamento de requisitos que envolveu a comunidade ns-3, que participou ativamente nesta fase ilustrando algumas das funcionalidades que pretendiam ver implementadas neste novo modelo de mobilidade.

Este projeto possui, também, uma página dedicada, na *wiki* do ns-3 [47], bem como uma página de arquivo detalhando as várias fases do projeto [48].

7.1.1.1 Requisitos funcionais

Os requisitos funcionais determinam as funcionalidades que devem ser integradas no *software* a desenvolver. Este trabalho procurou cumprir ao máximo os requisitos funcionais definidos inicialmente, bem como requisitos sugeridos pela comunidade e pelos tutores, ao longo do desenvolvimento do projeto, que serão enumerados de seguida.

Geração de mobilidade para um nó:

O utilizador deve ser capaz de gerar mobilidade para um nó especificando um ponto de partida e de chegada real. A mobilidade gerada deve refletir uma viagem usando a rede de estradas existente, à velocidade apropriada para cada troço do percurso.

Geração de mobilidade para um NodeContainer:

O utilizador deve ser capaz de gerar mobilidade para todos os nós contidos num NodeContainer, especificando uma área real de onde devem ser escolhidos os pontos de partida e os pontos de chegada. Para maior controlo, o utilizador deve ainda poder especificar um conjunto de localizações manualmente para serem usadas como pontos de partida e de chegada. A mobilidade gerada deve refletir viagens para todos os nós usando a rede de estradas existente, com velocidades apropriadas para cada troço do percurso.

Geração de mobilidade para os nós usando diferentes métodos de transporte:

Deve ser possível ao utilizador especificar o modo de transporte a usar, e a mobilidade gerada deve refletir as características desse método de transporte, bem como aderir à rede de transportes públicos ou de estradas existente. A velocidade dos nós deve, também, refletir as características do método de transporte selecionado.

Testes unitários e exemplos:

O módulo deve ainda possuir testes unitários para garantir o desempenho apropriado do mesmo, bem como alguns exemplos que ilustrem o seu funcionamento e uso.

***Helper*:**

Deve ser desenvolvido uma classe de *Helper*, que ofereça ao utilizador a API para este gerar e configurar a mobilidade para os seus nós. Este *Helper* deve conter funções que sejam capazes de realizar todos os casos de uso do módulo.

7.1.1.2 Requisitos não funcionais

Os requisitos não funcionais são propriedades ou características inerentes ao sistema, que os utilizadores necessitam e que afetam o nível de satisfação com o sistema desenvolvido. Também definem restrições que afetam o desenvolvimento da solução final [49].

Usabilidade:

O módulo deve ser de fácil uso para o utilizador e este deve, sem grande esforço, ser capaz de configurar e gerar mobilidade.

Implementação:

O módulo deve ser desenvolvido na linguagem C++, uma vez que é a linguagem utilizada no desenvolvimento dos módulos do ns-3. O módulo deve, ainda, ser integrado com as classes de mobilidade já existentes, bem como respeitar normas de codificação especificadas pelo consórcio ns-3.

Adaptação a diferentes fornecedores de serviços:

Deve ser possível implementar diferentes serviços de planeamento de viagens, bem como diferentes bases de dados de locais reais, sem alterações significativas ao código existente.

Realismo:

O módulo deve possuir um grau de realismo adequado, que deve ser posteriormente medido e avaliado.

7.1.2 Modelação e implementação

Nesta secção é apresentada a abordagem escolhida para a implementação do RoutesMobilityModel, uma parte integrante deste projeto. É também descrita a forma como as novas classes implementadas interagem com o simulador.

7.1.2.1 Modelação

O modelo de mobilidade descrito neste capítulo faz uso de serviços de planeamento de viagens externos de forma a calcular o melhor percurso entre dois pontos reais. A informação obtida a partir destes serviços é posteriormente descodificada e convertida em `Waypoints`, que são então adicionados ao `RandomWaypointMobilityModel` para serem processados.

Como tal, e após um estudo das opções existentes, foram escolhidos os serviços do *Google Maps*, descritos na secção 6 deste relatório. A escolha recaiu sobre estes serviços uma vez que,

para além de possuírem excelentes perspetivas de futuro, estes são robustos, bem documentados e suportados. Foram usadas duas APIs em particular, a *Directions API* e a *Places API.* A *Directions API* é central no funcionamento do módulo, uma vez que é esta API que retorna os pontos reais que constituem um trajeto entre dois locais reais. A API retorna um XML, contendo a informação relevante para o trajeto selecionado. A resposta da API agrupa a informação através dos seguintes elementos:

- *Leg* – Uma *Leg* é um conjunto de *Steps*. Num trajeto simples, existe apenas uma *Leg*. Contudo, em trajetos com paragens intermédias, existem várias *Legs,* uma para cada parte do percurso.
- *Step* – Um *Step* contém os pontos de uma secção do percurso codificados numa polilinha. Contém, também, o tempo estimado para aquela secção do percurso.

Esta informação, após tratamento, devolve as coordenadas geográficas de todos os pontos do trajeto. As coordenadas geográficas dos pontos são convertidas para coordenadas cartesianas, recorrendo, para isso, ao uso da biblioteca GeographicLib [50]. É ainda calculado o tempo estimado entre dois pontos recorrendo, para isso, ao tempo total estimado para aquele *Step*, estimado pelo serviço.

Desta forma, é possível criar `Waypoints` e, através do seu uso em conjunto com o `WaypointMobilityModel`, é possível gerar uma mobilidade que reflete o percurso real de um veículo ou um pedestre, utilizando variados métodos de transporte.

Uma vez que as simulações exigem muitas vezes números elevados de nós (centenas, milhares), foi necessário elaborar um método de geração de mobilidade para todos os nós de um `NodeContainer`. Para responder a este problema, foi usada a *Places API*, usando, assim, a sua base de dados de locais como pontos de origem e de chegada para os nós. Foi também elaborado um algoritmo que especifica uma área real, sendo gerados aleatoriamente, dentro dessa área, os pontos de origem e de chegada para os nós.

Atualmente, o modulo suporta apenas o uso das APIs do *Google Maps*, contudo, adicionar novas APIs e serviços requererá poucas alterações ao código existente, uma vez que foi usado o padrão *Strategy* para a implementação de ambas as classes que efetuam os pedidos às respetivas APIs.

A Figura 16 ilustra as interações entre os diferentes componentes do módulo, mostrando o seu funcionamento quando é pretendida a geração de mobilidade para um só nó.

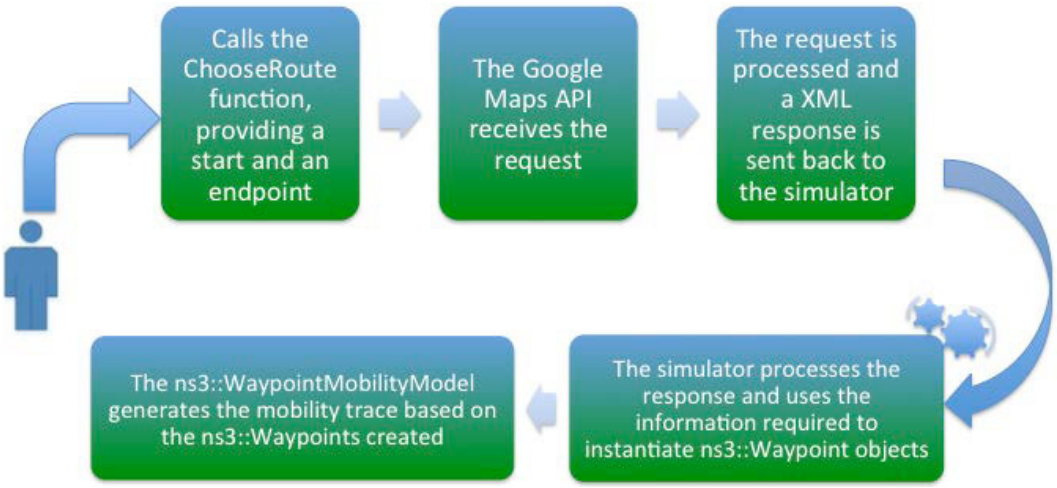


Figura 16 - Visão geral do funcionamento do módulo

7.1.2.2 Casos de Uso

De uma forma geral, o utilizador deste módulo é capaz de "Gerar mobilidade para um só nó", "Gerar mobilidade para um contentor de nós", "Gerar mobilidade para um nó com redirecionamento dinâmico" e "Gerar mobilidade através de ficheiros XML locais".

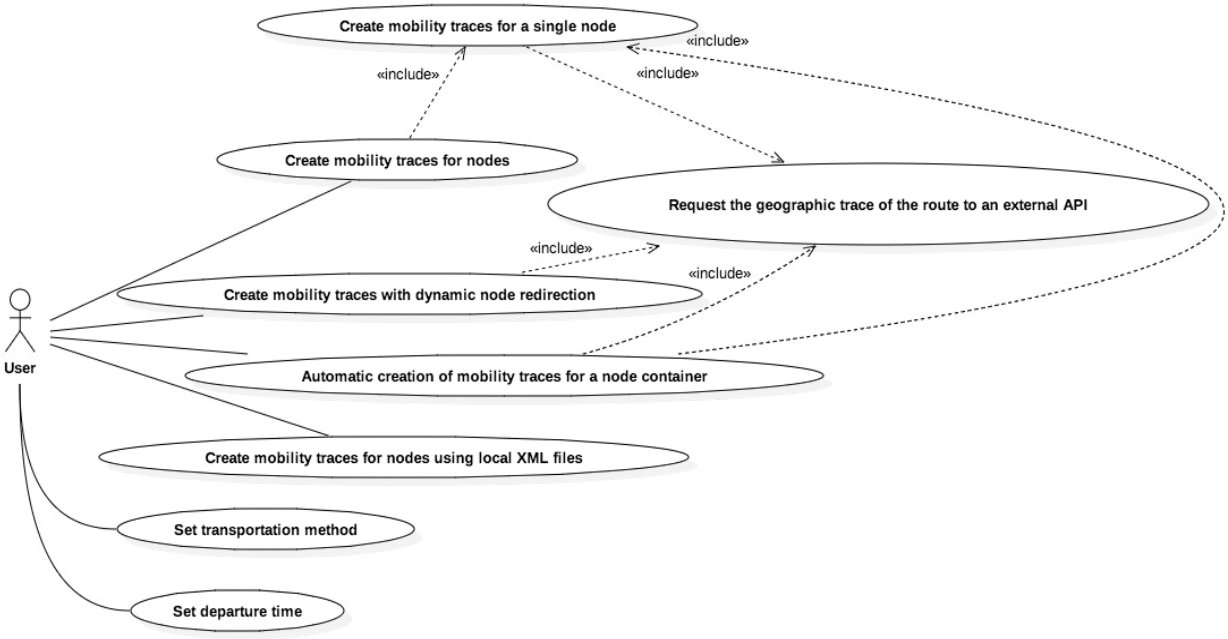


Figura 17 - Diagrama de casos de uso

A Figura 17 representa os casos de uso disponíveis a um utilizador deste módulo, que serão descritos em maior detalhe de seguida.

Geração de mobilidade para um nó

Este caso de uso é o mais simples dos desenvolvidos para deste módulo, permitindo ao utilizador gerar mobilidade para um nó, fornecendo para isso um ponto de partida e um ponto de chegada, bem como o nó para o qual deve ser gerada a mobilidade. Como pré-condições, o utilizador necessita de possuir acesso à internet e uma chave válida para a *Directions API*. O utilizador necessita ainda de especificar um ponto de partida e de chegada reconhecível pela *Directions API*. A Tabela 1 mostra o fluxo de eventos para este caso de uso.

ID	1	
Fluxo de Eventos		
Fluxo Principal	Ações do ator	Ações do sistema
	1. O utilizador pretende gerar mobilidade para um nó, fornecendo o ponto de partida e de chegada.	
		1. O sistema efetua o pedido à API
		2. O sistema retira da resposta da API a informação das *Legs* e dos *Steps* presentes na resposta, em particular, a polilinha de cada *Step*, bem como o tempo expectável para percorrer cada *Step*.
		3. O sistema converte a informação contida nas polilinhas em coordenadas geográficas, obtendo, assim, todos os Pontos da viagem.
		4. As coordenadas geográficas de cada Ponto são convertidas em coordenadas cartesianas pelo sistema
		5. O sistema calcula a distância total percorrida em cada *Step* e calcula o tempo em que cada Ponto deve ser atingido
		6. O sistema cria, então, *Waypoints* e adiciona-os ao `WaypointMobilityModel`

Tabela 1 – Fluxo de eventos principal para o caso de uso "Geração de mobilidade para um nó"

Geração de mobilidade para vários nós

Este caso de uso permite ao utilizador gerar mobilidade para vários nós, contidos num `NodeContainer`, mediante o fornecimento, por parte do utilizador do módulo, de um conjunto de locais, que serão escolhidos, aleatoriamente, pelo sistema, como pontos de início e de chegada.

O utilizador necessita de especificar um conjunto de locais, que a *Directions API* seja capaz de reconhecer corretamente, bem como de possuir uma chave válida para a API e uma ligação à internet. O fluxo de eventos principal pode ser visualizado na Tabela 2.

ID	2	
Fluxo de Eventos		
Fluxo Principal	Ações do ator	Ações do sistema
	1. O utilizador pretende gerar mobilidade para vários nós, fornecendo, manualmente, localizações reais.	
		1. O sistema escolhe, a partir do vetor de localizações fornecido, um ponto de partida e de chegada, aleatoriamente.
		2. O sistema efetua o pedido à API
		3. O sistema retira da resposta da API a informação das *Legs* e dos *Steps* presentes na resposta, em particular, a polilinha de cada *Step*, bem como o tempo expectável para percorrer cada *Step*.
		4. O sistema converte a informação contida nas polilinhas em coordenadas geográficas, obtendo, assim, todos os Pontos da viagem.
		5. As coordenadas geográficas de cada Ponto são convertidas em coordenadas cartesianas pelo sistema
		6. O sistema calcula a distância total percorrida em cada *Step* e calcula o tempo em que cada Ponto deve ser atingido
		7. O sistema cria, então, *Waypoints* e adiciona-os ao `WaypointMobilityModel`
		8. O sistema repete os passos **1** até **7** para todos os nós presentes no `NodeContainer`.

Tabela 2 - Fluxo de eventos principal para o caso de uso "Geração de mobilidade para vários nós"

Geração de mobilidade para um nó com redireccionamento dinâmico:

Este caso de uso permite ao utilizador gerar mobilidade para um nó, especificando uma localização para o qual deve ser redirecionado, bem como o tempo em que isso deve acontecer.

Para tal, o utilizador necessita de especificar um total de três locais, capazes de serem reconhecidos pela *Directions API*. Estes locais irão servir como pontos de partida e de destino originais. O terceiro local será a localização para o qual o utilizador deve ser redirecionado. O utilizador deve ainda especificar um valor temporal superior a zero.

A Tabela 3 mostra o fluxo de eventos principal para este caso de uso.

ID	3	
Fluxo de Eventos		
Fluxo Principal	Ações do ator	Ações do sistema
	1. O utilizador pretende gerar mobilidade com redirecionamento dinâmico, fornecendo as localizações originais de partida e de destino, bem como a localização para a qual o nó deve ser redirecionado e o tempo a que o redirecionamento deve ocorrer.	
		1. O sistema efetua o pedido à API
		2. O sistema retira da resposta da API a informação das *Legs* e dos *Steps* presentes na resposta, em particular, a polilinha de cada *Step*, bem como o tempo expectável para percorrer cada *Step*.
		3. O sistema converte a informação contida nas polilinhas em coordenadas geográficas, obtendo, assim, todos os Pontos da viagem.
		4. As coordenadas geográficas de cada Ponto são convertidas em coordenadas cartesianas pelo sistema
		5. O sistema calcula a distância total percorrida em cada *Step* e calcula o tempo em que cada Ponto deve ser atingido
		6. O sistema cria, então, os *Waypoints* cujo tempo é inferior ao tempo de redireccionamento e adiciona-os ao `WaypointMobilityModel`.
		7. O sistema verifica a localização geográfica do último ponto e efetua um pedido à API usando essa localização como ponto de partida e a localização de redirecionamento como ponto de destino.

		8. O sistema repete os passos **2** a **5**.
		9. O sistema cria, então, os *Waypoints*, adicionando-os ao `WaypointMobilityModel`.

Tabela 3 - Fluxo de eventos principal para o caso de uso "Geração de mobilidade para um nó com redirecionamento dinâmico

Geração automática de mobilidade para um contentor de nós:

Este caso de uso permite ao utilizador gerar mobilidade para um contentor de nós, sendo as localizações usadas como pontos de partida e de chegada escolhidas aleatoriamente pelo sistema, a partir de um área definida pelo utilizador. De forma a obter uma geração de mobilidade com sucesso neste caso de uso, o utilizador necessita de fornecer coordenadas validas, para a área. O utilizador necessita ainda de possuir uma ligação à internet e uma chave válida para as APIs a usar.

ID	4	
Fluxo de Eventos		
Fluxo Principal	Ações do ator	Ações do sistema
	1. O utilizador pretende gerar mobilidade para todos os nós de um `NodeContainer` automaticamente, fornecendo a área onde deve ser gerada essa mobilidade.	
		1. O sistema escolhe coordenadas, aleatoriamente, contidas na área especificada.
		2. O sistema efetua o pedido à API.
		3. O sistema retira da resposta da API a informação das *Legs* e dos *Steps* presentes na resposta, em particular, a polilinha de cada *Step*, bem como o tempo expectável para percorrer cada *Step*.
		4. O sistema converte a informação contida nas polilinhas em coordenadas geográficas, obtendo, assim, todos os Pontos da viagem.
		5. As coordenadas geográficas de cada Ponto são convertidas em coordenadas cartesianas pelo sistema
		6. O sistema calcula a distância total percorrida em cada *Step* e calcula o tempo em que cada Ponto deve ser atingido

		7. O sistema cria, então, os *Waypoints* cujo tempo é inferior ao tempo de redireccionamento e adiciona-os ao `WaypointMobilityModel`.
		8. O sistema repete os passos **1** a **7** para todos os nós do `NodeContainer`.

Tabela 4 - Fluxo de eventos principal para o caso de uso "Geração automática de mobilidade para um contentor de nós"

Geração de mobilidade através de ficheiros XML locais:

Durante a fase de análise do problema, em conjunto com os mentores do projeto, foi determinado que deveria ser possível ao utilizador gerar mobilidade sem ligação à *internet*, uma vez que, atualmente, o ns-3 não depende do acesso à *internet* para ser usado.

Este caso de uso permite ao utilizador descarregar, manualmente, as respostas da API e carrega-las, posteriormente, para o simulador. Para tal, o utilizador necessita de especificar o caminho para o ficheiro ou diretório onde se encontra a mobilidade. É necessário, ainda, que os ficheiros XML sejam validos.

ID	5	
Fluxo de Eventos		
Fluxo Principal	Ações do ator	Ações do sistema
	1. O utilizador pretende gerar mobilidade a partir de ficheiros XML previamente descarregados	
		1. O sistema abre o ficheiro XML descarregado para leitura
		2. O sistema retira do ficheiro XML a informação das *Legs* e dos *Steps* presentes na resposta, em particular, a polilinha de cada *Step*, bem como o tempo expectável para percorrer cada *Step*.
		3. O sistema converte a informação contida nas polilinhas em coordenadas geográficas, obtendo, assim, todos os Pontos da viagem.
		4. As coordenadas geográficas de cada Ponto são convertidas em coordenadas cartesianas pelo sistema
		5. O sistema calcula a distância total percorrida em cada *Step* e calcula o tempo em que cada Ponto deve ser atingido
		6. O sistema cria, então, os *Waypoints* cujo tempo é inferior ao tempo de

		redireccionamento e adiciona-os ao `WaypointMobilityModel`.
		7. O sistema repete os passos **1** a **6** para todos os nós do `NodeContainer`.

Tabela 5 - Fluxo de eventos principal para o caso de uso "Geração de mobilidade através de ficheiros XML locais"

7.1.2.3 Classes implementadas

No início do desenvolvimento deste módulo foi efetuado um levantamento das classes necessárias para a sua implementação, bem como a sua integração com as classes já existentes no ns-3, nomeadamente as referidas na secção 5.7.3.

Tal como referido anteriormente, existe uma grande integração deste modulo com o, já existente, `WaypointMobilityModel`. Este modelo de mobilidade é o ideal para usar em conjunto com o módulo desenvolvido, uma vez que já possui as ferramentas básicas para o desenvolvimento do mesmo. Desta forma, o módulo beneficia ainda da robustez oferecida pelo `WaypointMobilityModel`.

Foi, também, imprescindível o desenvolvimento de novas classes, de interface com a API que permita extrair a informação e trata-la. Estas classes são centrais ao desenvolvimento deste módulo, uma vez que é nelas que estão implementadas as heurísticas que lhe conferem o seu funcionamento e realismo. Foram, ainda, implementadas classes que representam as *Leg, Step* e Pontos retornados pela API. Estas classes representam a informação obtida, após tratamento.

Uma vez que a API do *Google Maps* retorna um ficheiro XML foi, ainda, necessário implementar um *parser* de XML. Durante a fase de análise do problema foi escolhido um *parser* do tipo SAX2. Este tipo de *parser* é orientado a eventos, caracterizando-se assim, por um baixo consumo de memória[51], [52], sendo, então, a melhor escolha para a recolha de informações do XML, uma vez que apenas alguns parâmetros da resposta são aproveitados. Foi, então, escolhida a biblioteca Xerces-C++[53] para efetuar o *parse* da resposta XML, sendo necessário implementar ainda dois *handlers*, que recuperam a informação das respostas da *Directions API* e da *Places API*.

Por fim, foi ainda criado um assistente para fornecer ao utilizador todas as funcionalidades do módulo. Este assistente visa, assim, permitir uma fácil geração e configuração de todos os aspetos disponíveis da mobilidade a gerar.

De forma a validar o funcionamento do módulo, foi ainda implementado um teste unitário, que verifica os algoritmos de conversão entre coordenadas geográficas e cartesianas, bem como o algoritmo que calcula o tempo em que um `Waypoint` deve ser atingido pelo nó. Foram ainda implementados três exemplos que ilustram o funcionamento básico do módulo.

A Figura 18 ilustra a arquitetura desenvolvida para este módulo, bem como a integração do módulo com o código do ns-3 já existente. Os detalhes de cada classe implementada, bem como os exemplos e testes, serão descritos em mais detalhe numa próxima secção deste relatório. O diagrama de classes completo encontra-se no Anexo 2 este relatório.

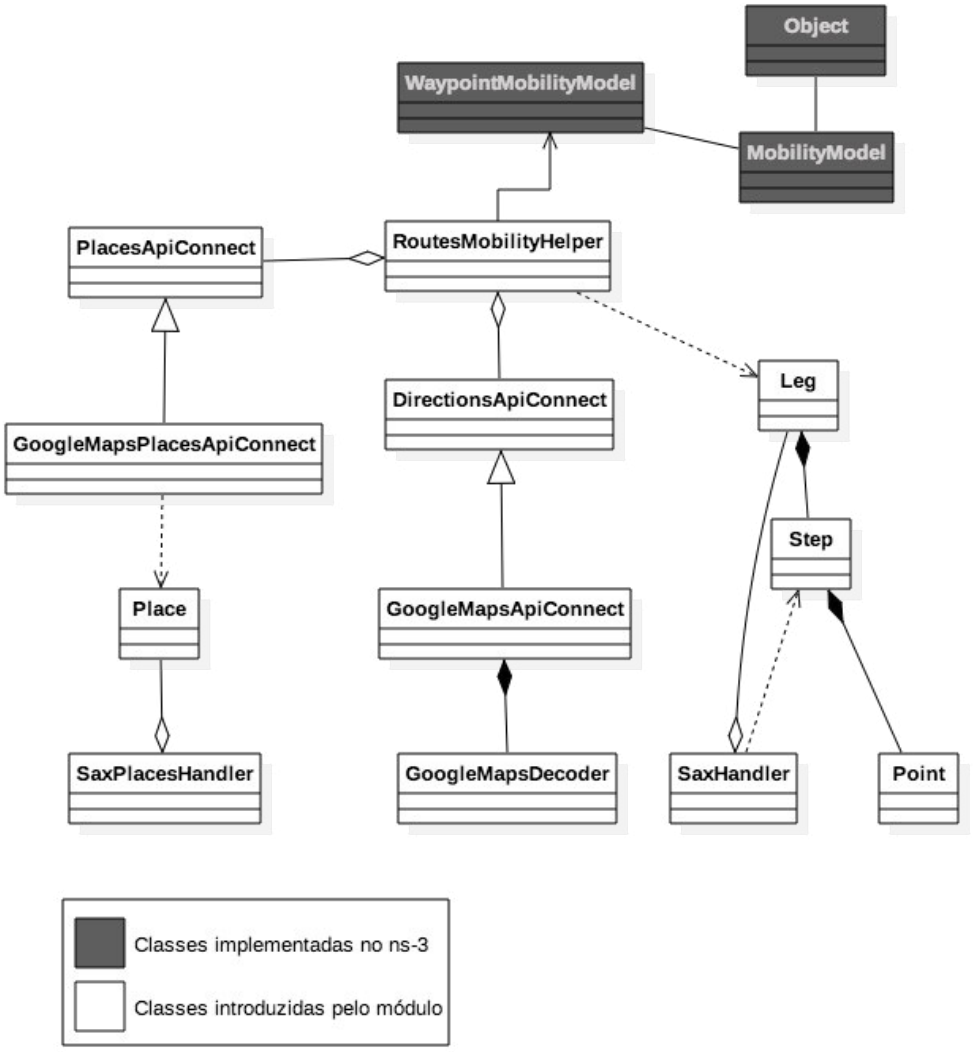


Figura 18 - Diagrama de classes simplificado do RoutesMobilityModel

A classe `GoogleMapsApiConnect` (Figura 20) contem as funções necessárias à ligação à Directions API. Esta classe implementa a interface `DirectionsApiConnect` (Figura 19), que deve ser implementada por todas as classes que pretendam estabelecer uma ligação a um serviço externo de planeamento de rotas.

A classe `GoogleMapsDecoder` (Figura 21) implementa as funções responsáveis por descodificar a polilinha, por calcular o tempo em que os `Waypoints` devem ser atingidos e, ainda, por converter as coordenadas geográficas em coordenadas cartesianas.

As classes `Leg`, `Step`, `Point` e `Place` (Figura 22, Figura 23, Figura 24 e Figura 25, respetivamente) representam o modelo conceptual desta arquitetura. Cada classe `Step` contem uma lista com `Points` e cada classe `Leg` contem, por sua vez, uma lista com `Steps`. A classe `Place` define um local real

O *parser* de XML requer a definição de *handlers*, responsáveis por gerir a leitura da informação obtida a partir da resposta de uma dada API. Estes *handlers* estão implementados nas classes `SaxHandler` (Figura 28) e `SaxPlacesHandler` (Figura 29), que contem, assim, as funções necessárias à análise da informação das APIs *Directions* e *Places*, respetivamente.

A classe `GoogleMapsPlacesApiConnect` (Figura 27) define, por sua vez, o comportamento necessário para efetuar a ligação à *Places API.* Esta classe implementa a interface `PlacesApiConnect` (Figura 26), que deve ser implementada por todas as classes que pretendam efetuar uma ligação a uma base de dados de locais reais.

Por fim, a classe `RoutesMobilityHelper` (Figura 30) define um assistente que permite ao utilizador gerar e configurar a mobilidade pretendida.

Estas classes serão discutidas em mais detalhe de seguida, com a descrição de todas as classes desenvolvidas, apresentando os seus atributos e funções.

As classes `Object`, `MobilityModel` e `WaypointMobilityModel` são classes já existentes no ns-3 e não foram feitas alterações a estas classes no âmbito do desenvolvimento deste módulo.

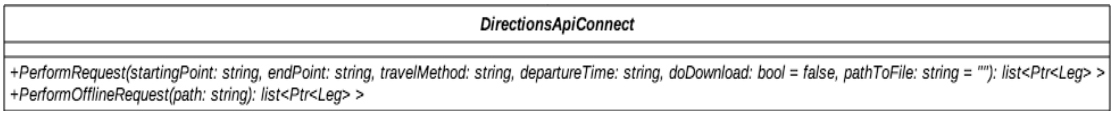


Figura 19 - Classe DirectionsApiConnect

`DirectionsApiConnect`

Esta interface define o comportamento a implementar nas classes cujo propósito é efetuar um pedido a um serviço de planeamento de rotas. Possui a assinatura de duas funções, `list<Ptr<Leg> > PerformRequest(string startingPoint, string endPoint, string travelMethod, string departureTime, bool`

doDownload, string pathToFile) e list<Ptr<Leg> > PerformOfflineRequest(string path).

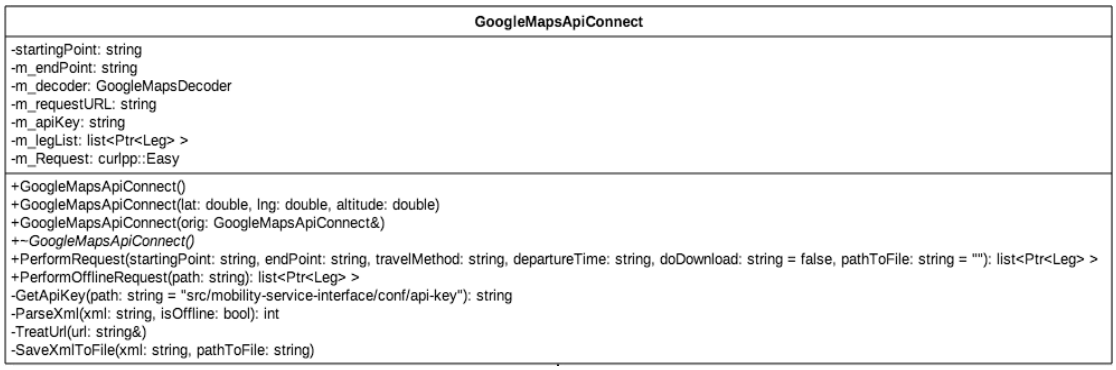


Figura 20 - Classe GoogleMapsApiConnect

GoogleMapsApiConnect

Esta classe implementa a interface DirectionsApiConnect, desta forma definindo as funções PerformRequest e PerformOfflineRequest. A função PerformRequest é a função responsável por efetuar o pedido à API, com os diferentes parâmetros aplicados. A resposta da API é também processada, pela função ParseXml(string xml, bool isOffline), que efetua o *parse* da resposta, bem como uma chamada à função ConvertToCartesian, da classe GoogleMapsDecoder, que será discutida mais à frente neste relatório.

A função PerformOfflineRequest é semelhante à PerformRequest, no sentido em que permite ao módulo carregar uma resposta da API, previamente descarregada pelo utilizador, e efetuar o seu tratamento, através da função ParseXml.

Esta classe contém, ainda, as funções TreatUrl(string& url) e GetApiKey(string path), que efetuam algumas tarefas simples mas necessárias ao bom funcionamento do módulo. A função TreatUrl é responsável por corrigir alguns erros de má formação do URL, uma vez que algumas porções do URL são especificadas pelo utilizador, tais como o ponto de partida e de chegada. Uma vez que o utilizador pode inserir caracteres não permitidos num URL (um espaço, por exemplo), tais como definidos no RFC 3986 [54], esta função é responsável por tratar o URL de forma a converter todos os caracteres não autorizados para o seu equivalente em *percent-encoding.*

A função GetApiKey é a função responsável por ler a chave da API que o utilizador deve colocar num ficheiro de texto. O utilizador pode especificar um caminho para este ficheiro ou optar pelo uso do caminho aconselhado, na pasta conf do módulo.

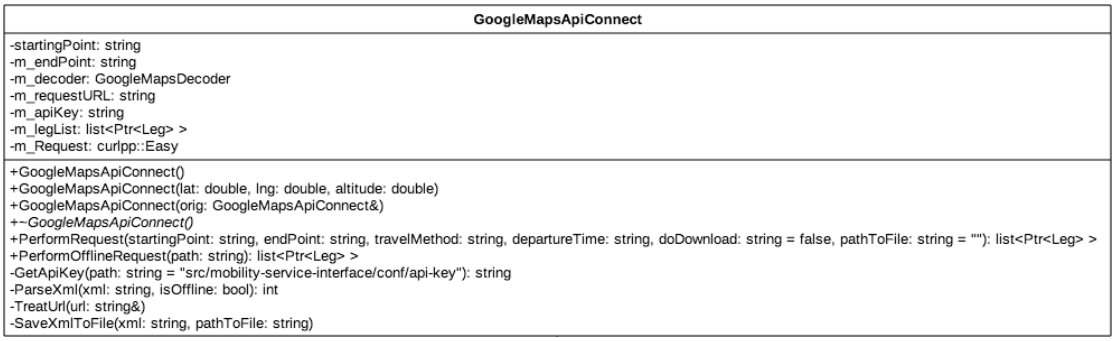


Figura 21 - Classe GoogleMapsDecoder

`GoogleMapsDecoder`

Esta classe é uma classe utilitária, responsável por efetuar muito do tratamento da informação obtida a partir da *Directions API*.

A função `ConvertToCartesian(LegList& leg)` recebe por parâmetro uma lista de *Legs* com a informação sobre todos os *Steps* do percurso. Cada *Step* contem uma polilinha, que é posteriormente descodificada através da função `ConvertToGeoCoordinates(string polyline, list<Ptr<Point> >& pointList)`, preenchendo, assim, a lista de pontos passada por parâmetro com as coordenadas geográficas dos pontos contidos nessa polilinha. De seguida são convertidas as coordenadas geográficas desses pontos para coordenadas cartesianas. Por fim, é também calculado o tempo em que cada ponto deve ser atingido, através da função `FillInWaypointTime`.

Cada `Waypoint` necessita de ser instanciado no espaço e no tempo. Desta forma, quando o tempo de simulação é igual ao de um dado `Waypoint`, o nó encontra-se na posição indicada por esse `Waypoint`. A função `FillInWaypointTime (list<Ptr<Point>>,double,list<Ptr<Step>::iterator,list<Ptr<Step>>::iterator,list<Ptr<Leg>>::iterator,list<Ptr<Leg>>,double)` é responsável por preencher o tempo dos pontos de um dado `Step`. Uma vez que não é conhecido o tempo em que cada ponto deve ser atingido durante o percurso, é necessário computar este valor, recorrendo para isso ao tempo que a Google determina como o necessário para percorrer o *Step* na totalidade, que nesta função é representado pela variável `travelTime`. A função recebe ainda a distância total percorrida num dado *Step*, computada através de uma chamada à função `CalculateTotalDistanceInStep` da classe `Leg`, que será explicada mais à frente neste relatório. A função `SetWaypointTime (double startAt, double totalDistanceInStep, Ptr<Point> p1, Ptr<Point> p2, double travelTime)` é invocada pela função `FillInWaypointTime` e é

responsável por fazer os cálculos de espaçamento temporal dos `Points`, computando, para tal, a distância parcial entre dois `Point`, recorrendo à função `CalculatePartialDistance`, e, usando a distância total percorrida e o tempo total esperado para terminar aquele *Step*, é possível estimar o tempo de cada `Point` de um dado `Step`, desta forma espaçando os pontos no tempo proporcionalmente à distância percorrida. Uma vez que podem existir dezenas de pontos numa curta distância, isto permite ao módulo introduzir variações de velocidade durante o percurso. Estas, na prática, traduzem-se em obstáculos rodoviários, tais como rotundas e cruzamentos, por exemplo, e na alteração das características da estrada a percorrer naquele momento, tal como as acelerações ou desacelerações ocorridas ao entrar ou sair de uma autoestrada, entre outras.

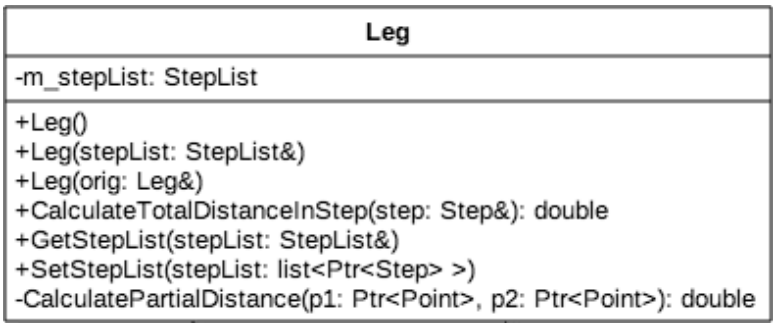


Figura 22 - Classe Leg

Leg

Esta classe representa o conceito homónimo introduzido pela *Directions API*. Como tal, esta classe contem uma lista de `Steps`, que caracterizam as diferentes secções da mesma.

A classe possui alguns métodos utilitários, tais como o `GetStepList` e `SetStepList`, que permite obter e atribuir uma lista de steps a esta ler, respetivamente. Possui ainda um método `CalculateTotalDistanceInStep(Step& step)`, que permite calcular o total da distância percorrida num dado `Step`.

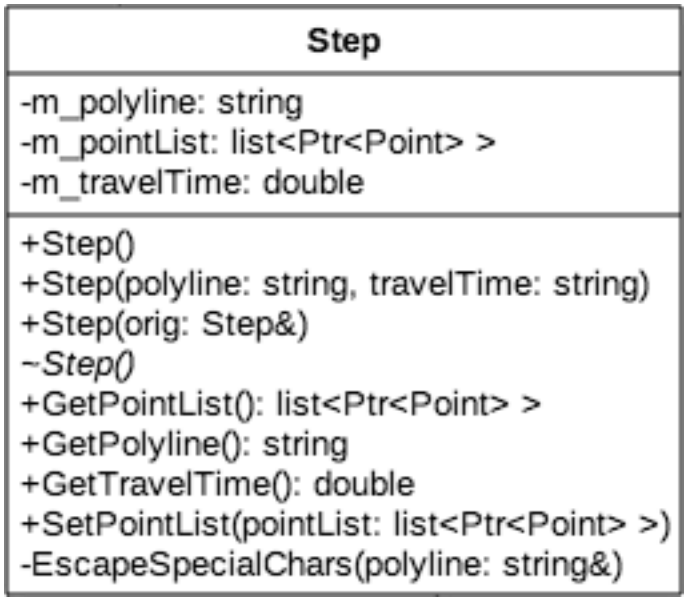


Figura 23 - Classe Step

`Step`

Esta classe representa, também, o conceito homónimo obtido na resposta da *Directions API*. Possui três atributos, uma *string* que contem a polilinha (`m_polyline`), um *double* que representa o tempo de total de viagem de um dado *Step* (`m_travelTime`), obtidos a partir da *Directions API,* e ainda uma lista de `Points`.

A função `EscapeSpecialChars(string& polyline)` é essencial ao bom funcionamento do módulo, uma vez que é responsável por tratar os caracteres '\'. Estes caracteres, quando associados a outros caracteres, como o '\t' ou '\n', são interpretados como um só e, uma vez que o algoritmo de conversão da polilinha faz uso do código ASCII de cada carácter, um simples carácter errado é o suficiente para fornecer resultados inesperados.

Possui ainda algumas funções utilitárias, tais como `GetPointList()`, `GetPolyline()`, `GetTravelTime ou SetPointList`, que permitem aceder aos atributos desta classe.

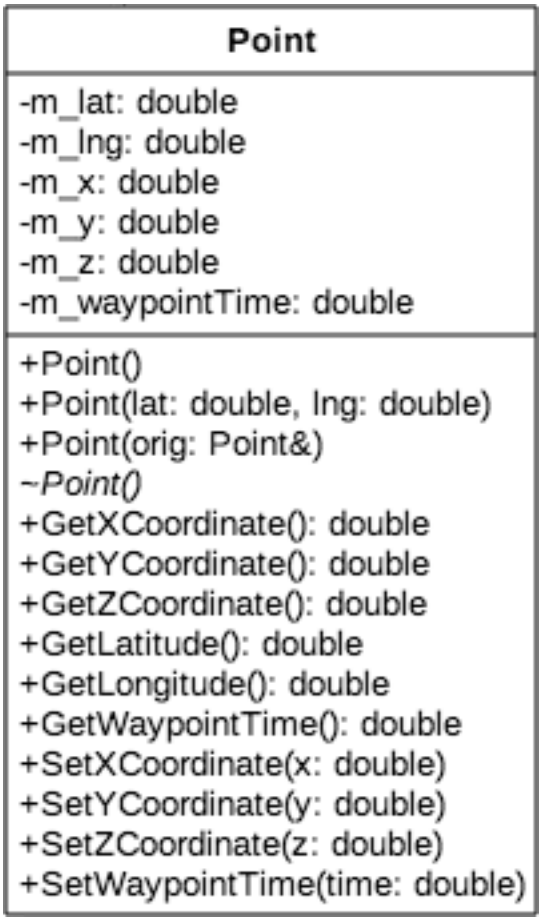


Figura 24 - Classe Point

`Point`

Esta classe representa o conceito de pontos, que não existe na resposta da *Directions API*. O objetivo desta classe é a de reunir todas as informações acerca de um dado ponto do trajeto.

Como tal, possui como atributos a latitude e longitude (`m_lat` e `m_lng`), as coordenadas no plano cartesiano (`m_x`, `m_y` e `m_z`) e o tempo em que esse ponto deve ser atingido. Possui ainda funções *getters* e *setters* para permitir o acesso a essas variáveis

Os atributos latitude e longitude são obtidos através da descodificação direta da polilinha, que são depois usadas para a conversão para coordenadas do plano cartesiano. Por fim, o tempo é calculado, de forma a espaçar os pontos no tempo.

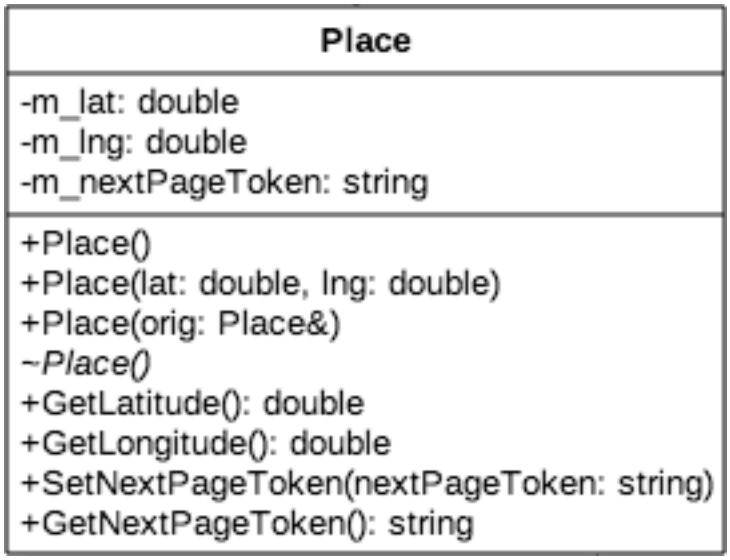


Figura 25 - Classe Place

`Place`

Esta classe representa o conceito de *Place* obtido a partir da *Places API*. Um *Place* é um local real (restaurantes, cafés, locais de interesse, etc), e possui a latitude e longitude (`m_lat` e `m_lng`). Possui ainda um atributo do tipo *string*, o `m_nextPageToken`. Este atributo é usado para pedir mais resultados à API, tal como explicado na secção 6.2 deste relatório.

Esta classe possui ainda as habituais funções *getters* e *setters*, de modo a permitir o acesso aos atributos mencionados.

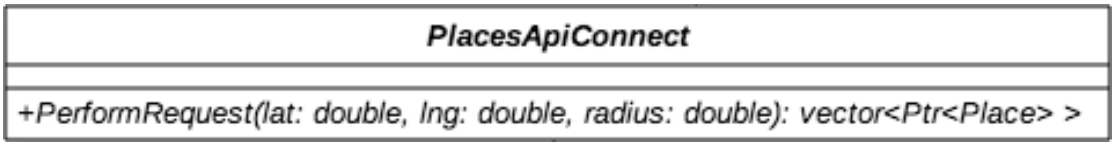


Figura 26 - Classe PlacesApiConnect

`PlacesApiConnect`

Esta interface define o comportamento a implementar nas classes cujo propósito é efetuar um pedido a uma base de dados de locais externa. Possui a assinatura de uma função, `PerformRequest(double lat, double lng, double radius)`.

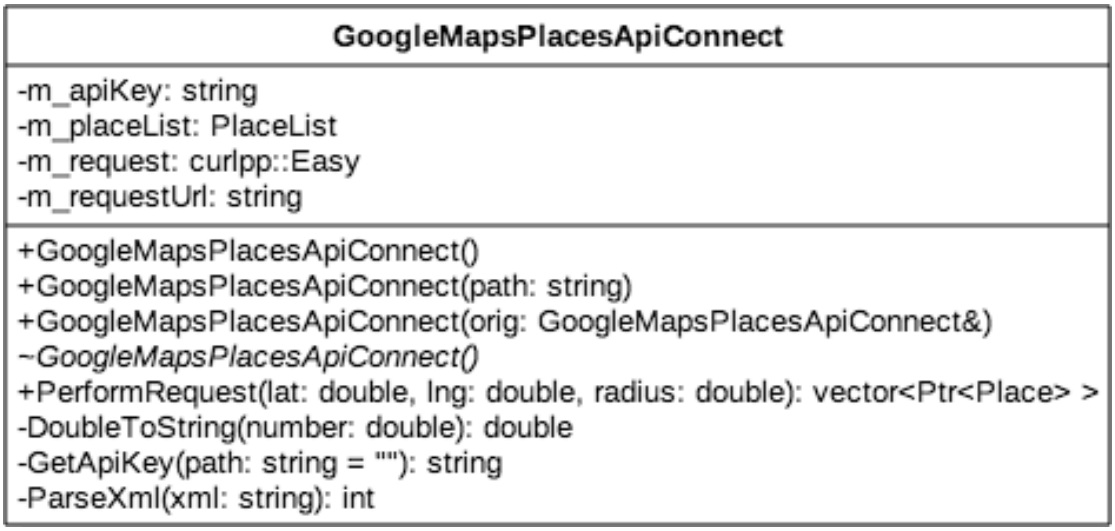


Figura 27 - Classe GoogleMapsPlacesApiConnect

`GoogleMapsPlacesApiConnect`

Esta classe implementa a interface `PlacesApiConnect`, implementando assim a função `PerformRequest`. Esta função recebe por parâmetro a latitude e longitude do centro da pesquisa e o raio máximo de pesquisa a partir do centro, que são necessários para efetuar o pedido de locais à *Places API*. Após efetuar o pedido é executada a função `ParseXml (string xml)`, que processa a resposta da API, criando objetos do tipo `Place`. Uma vez que pode ser necessário repetir o pedido à API com um `pagetoken` diferente, a função `ParseXml` é executada até que a API deixe de retornar o atributo `pagetoken`, desta forma obtendo a totalidade de *Places* retornados pela API.

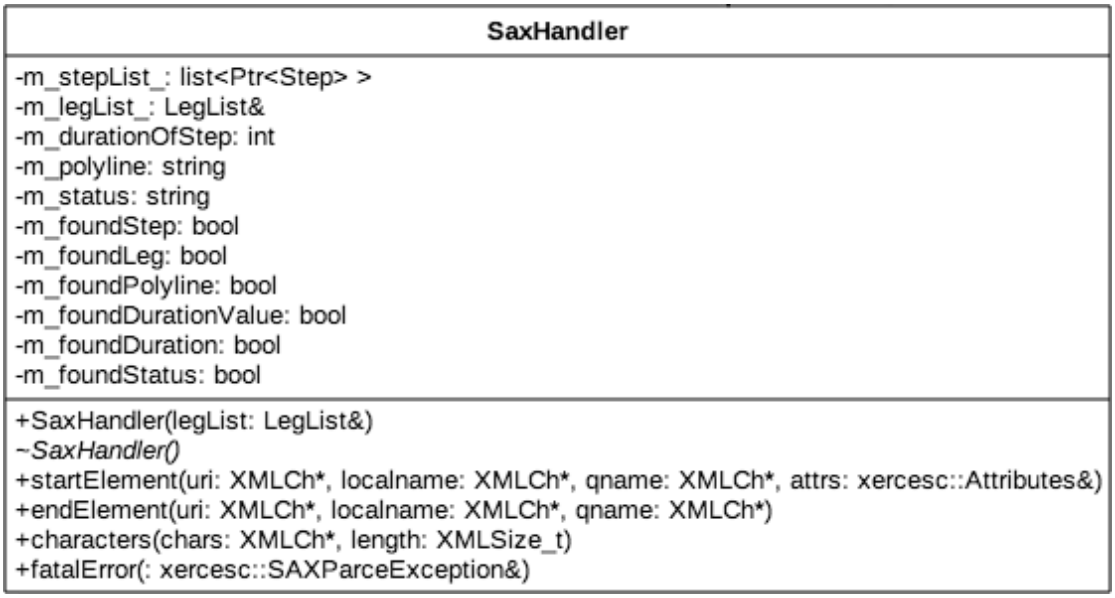


Figura 28 - Classe SaxHandler

`SaxHandler`

Esta classe é responsável por definir o comportamento do *parser* responsável por analisar as respostas provenientes da *Directions API*. Como tal, implementa a interface `xercesc::DefaultHandler`. Esta interface define algumas funções a implementar, tais como `startElement` e `endElement`. Sendo o SAX um *parser* baseado em eventos, estas funções definem a ação a realizar quando é encontrado um início de elemento e um fim de um elemento, respetivamente. A função `characters`, por sua vez, define a ação a tomar de forma a ler a informação contida num dado elemento.

O construtor desta classe recebe, por referência, a lista de `Legs`, onde irá guardar a informação obtida a partir do XML. Isto significa que os objetos do tipo `Leg` e `Step` são criados e adicionados à respetiva lista à medida que vão sendo encontrados no XML. Esta abordagem está de acordo com as melhores práticas[55] no uso deste tipo de *parser*, uma vez que apenas é possível aceder aos campos durante a leitura sequencial do XML.

Esta classe é, também, responsável por processar o estado da resposta da API, desta forma validando se ocorreu algum problema, tal como o facto de o utilizador ter esgotado a sua cota, por exemplo. Para isto, é analisado o campo `status`, presente na resposta da API, e qualquer valor diferente de `OK` ativa um *macro* `NS_ASSERT_MSG` que termina a simulação e fornece uma mensagem de erro ao utilizador, de acordo com o valor encontrado na variável de `status`.

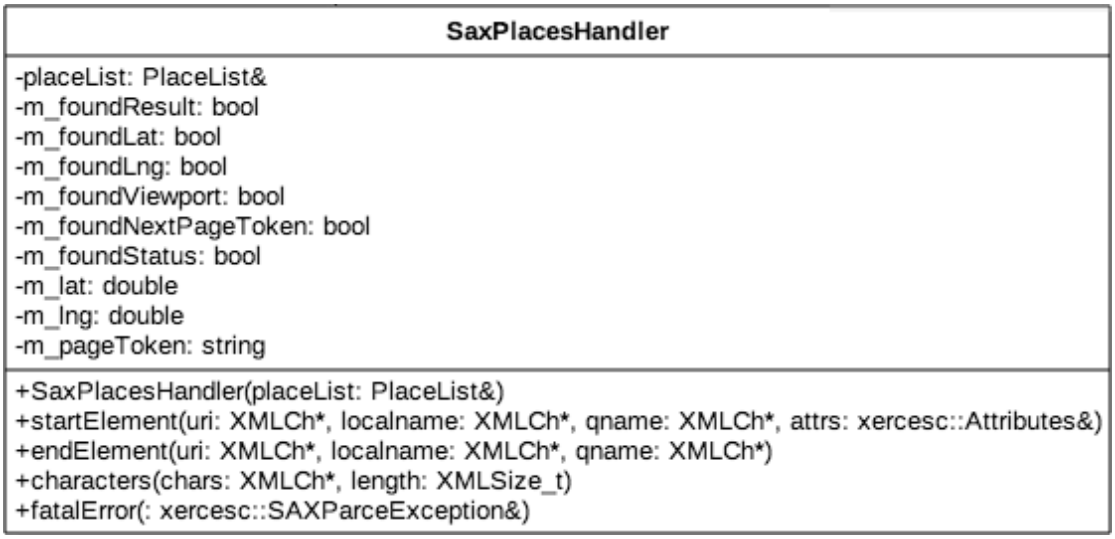


Figura 29 - Classe SaxPlacesHandler

SaxPlacesHandler

Esta classe, à semelhança da classe `SaxHandler`, implementa a interface `xercesc::DefaultHandler`, desta vez para analisar as respostas enviadas pela *Places API*. O construtor desta classe recebe, por referência, um *vector* de `Places` onde irá guardar a informação obtida a partir do XML, o que significa que esta classe é responsável por criar os objetos do tipo `Place` e adiciona-los à respetiva estrutura de dados.

À semelhança da classe descrita anteriormente, esta classe analisa, também, o valor do atributo `status`, terminando a simulação no caso de o valor ser diferente de `OK`, fornecendo uma mensagem de erro ao utilizador.

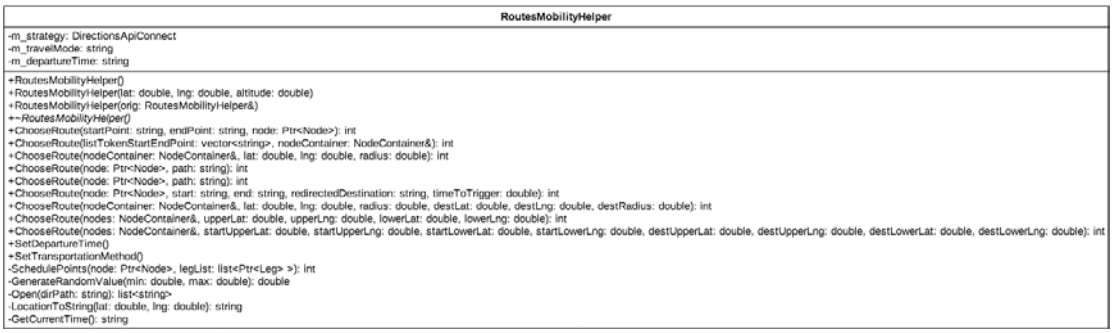


Figura 30 - Classe RoutesMobilityHelper

RoutesMobilityHelper

Esta classe é responsável por fornecer ao utilizador uma interface para configurar e usar o módulo. O construtor da classe recebe por parâmetro a latitude, longitude e altitude de um local geográfico que será usado, no plano cartesiano, como o centro do plano cartesiano. O local geográfico passado será, então, usado como referencia, pela GeographicLib, para a conversão entre coordenadas geográficas e coordenadas cartesianas.

A classe dispõe de algumas funções de configuração, tais como a função `SetTransportationMethod` e a função `SetDepartureTime`. A função `SetTransportationMethod` permite ao utilizador especificar o método de transporte sobre o qual deve ser gerada a mobilidade para os seus nós. Esta configuração é então passada à API, aquando da realização do pedido à mesma, que retorna a informação conhecida sobre o trajeto, mediante o método de transporte selecionado. A função `SetDepartureTime` permite ao utilizador especificar a data e a hora reais que a API deve usar para calcular o trajeto pretendido. Esta função é especialmente importante para controlar a mobilidade gerada quando é selecionado, como método de transporte, o transporte público. Esta função é, também, crucial no controlo da mobilidade gerada quando são pedidos trajetos que tenham

em conta as informações de trânsito para um dado percurso. Uma vez que o parâmetro é essencial para pedidos cujo método de transporte é o transporte público, por defeito, é selecionado o valor atual do relógio do sistema, para a variável `m_departureTime`, em EPOCH.

A classe descrita possui ainda algumas funções utilitárias, responsáveis por assistir na geração de mobilidade, como a função `GenerateRandomValue(double iMin, double iMax)` e a função `LocationToString(double lat, double lng)`, entre outras. A função `GenerateRandomValue` gera um número pseudoaleatório, recorrendo ao uso da função `rand`, dentro do intervalo especificado pelos parâmetros, enquanto que a função LocationToString recebe uma latitude e uma longitude que irá juntar numa *string*.

Esta classe é ainda responsável por fornecer ao utilizador as funções necessárias à geração da mobilidade, propriamente dita. Existem, para o efeito, um total de sete funções que permitem diferentes formas de geração de mobilidade, as funções `ChooseRoute`.

As funções `ChooseRoute`, devido à sua complexidade e ao facto de serem fundamentais na geração de mobilidade, serão explicadas de seguida, detalhadamente. Será, também, abordada em detalhe a função `SchedulePoints`, essencial para a geração de mobilidade.

`ChooseRoute(string startPoint, string endPoint, Ptr<Node> node)`

Esta função implementa o caso base de geração de mobilidade, gerando mobilidade para um único nó. O utilizador necessita apenas de passar à função, a partir do ambiente de simulação, um ponto de partida (`startPoint`) e um ponto de destino (`endPoint`), bem como o nó (`node`) a que deseja aplicar a mobilidade. Os pontos de partida e destino podem ser nomes de ruas, locais de interesse ou, até, coordenadas geográficas.

A função começa por invocar a função `PerformRequest`, explicada anteriormente, recebendo uma lista de `Legs` preenchida com toda a informação necessária para criar `Waypoints`. É, então, invocada a função `SchedulePoints`, responsável por criar e adicionar `Waypoints` ao `WaypointMobilityModel` e, desta forma, gerar a mobilidade propriamente dita. Esta função será abordada em maior detalhe numa próxima secção deste relatório.

`ChooseRoute(std::string const *listTokenStartEndPoint, NodeContainer &nodeContainer)`

Esta função recebe por argumento um vetor com um conjunto de localizações reais, especificadas pelo utilizador, bem como o `NodeContainer` onde se pretende que a mobilidade seja gerada. A função escolhe, aleatoriamente, como pontos de partida e de chegada localizações obtidas a partir do vetor passado pelo utilizador e passa-os à função `ChooseRoute` base.

Esta função, embora necessite de uma maior interação por parte do utilizador, tem o potencial de proporcionar um maior controlo da mobilidade gerada.

`ChooseRoute(NodeContainer &nodeContainer, double lat, double lng, double radius)`

Esta função recebe por parâmetro o `NodeContainer` para o qual deve ser gerada a mobilidade. Recebe, ainda, uma localização real, que a API usará como centro da pesquisa, e o raio dessa pesquisa.

Após o contacto com a *Places API*, é, então, obtido um *vector* que contem todas as localizações obtidas a partir da *Places API*. Estas localizações são, então, combinadas, aleatoriamente, como pontos de partida e de chegada e é invocada a função `ChooseRoute` base com essas mesmas localizações, de forma a obter a mobilidade pretendida.

`ChooseRoute(NodeContainer &nodeContainer, double lat, double lng, double radius, double destLat, double destLng, double destRadius)`

Esta função é semelhante, em funcionamento, à função descrita anteriormente. As duas funções diferem no facto de que esta função escolhe os locais de partida e de destino a partir de duas *pools* distintas. Desta forma, a mobilidade gerada tem início numa determinada área e um destino noutra área real diferente.

A função recebe, por parâmetro, o `NodeContainer` sobre o qual deve ser gerada a mobilidade, e duas localizações geográficas onde a *Places API* deve procurar locais. Como tal, são efetuados dois pedidos à API, sendo recebidos dois *vectors* distintos, contendo as localizações que devem ser usadas como pontos de partida e as localizações que devem ser usadas como pontos de destino.

`ChooseRoute(NodeContainer& nodes, double upperLat, double upperLng, double lowerLat, double lowerLng)`

Esta função é capaz de gerar mobilidade para um `NodeContainer`, usando coordenadas escolhidas aleatoriamente a partir de uma área especificada pelo utilizador. Como tal, esta função recebe por parâmetro as coordenadas máximas e mínimas para a área escolhida. De seguida são geradas coordenadas aleatórias contidas nessa área como pontos de partida e de destino para todos os nós.

Esta função é ideal para a geração de mobilidade para `NodeContainers` numerosos, uma vez que permite ao utilizador, com pouco esforço, selecionar pontos de partida e de chegada único para todos os seus nós a partir de uma área especificada.

`ChooseRoute(NodeContainer& nodes, double startUpperLat, double startUpperLng, double startLowerLat, double startLowerLng, double destUpperLat, double destUpperLng, double destLowerLat, double destLowerLng)`

Esta função é semelhante no funcionamento à função descrita anteriormente. As duas funções diferem no facto de esta função gera, aleatoriamente, os pontos de partida contidos numa dada área e os pontos de destino contidos numa outra área. O utilizador deve, então, especificar a latitude e longitude máxima e mínima para as duas áreas, bem como o `NodeContainer` para o qual deve ser gerada a mobilidade.

A mobilidade é gerada recorrendo à função `ChooseRoute` base, usando como pontos de partida e de destino as coordenadas geradas, desta forma forçando os nós a começarem o trajeto numa determinada área e a terminarem o seu trajeto numa outra área real.

`ChooseRoute(Ptr<Node> node, std::string path)`

Esta função permite ao utilizador gerar mobilidade para um nó a partir de uma resposta da API descarregada manualmente. Para tal, a função recebe o nó sobre o qual deve ser gerada a mobilidade e o caminho para o ficheiro XML. Recorrendo à função `PerformOfflineRequest`, explicada anteriormente, é possível tratar a resposta descarregada e obter uma lista de `Legs` que são de seguida passadas à função `SchedulePoints`, de forma a gerar a mobilidade propriamente dita.

`ChooseRoute(NodeContainer nodeContainer, string dirPath)`

Esta função é semelhante à descrita anteriormente, contudo carrega todas as respostas da *Directions API* presentes numa dada diretoria local, gerando assim mobilidade para todos os nós presentes no `NodeContainer`. Para tal, o utilizador fornece o caminho para a diretoria desejada, bem como o `NodeContainer` para o qual deve ser gerada a mobilidade. O utilizador deve especificar uma diretoria contendo ficheiros XML suficientes, pelo menos, para todos os nós do `NodeContainer`.

Esta função recorre, então, à função explicada anteriormente para gerar a mobilidade propriamente dita.

`ChooseRoute(Ptr<Node> node, std::string start, std::string end, std::string redirectedDestination, double timeToTrigger)`

Esta função permite ao utilizador gerar mobilidade para um nó, especificando um ponto de partida e um ponto de destino reais. No entanto, permite ainda redirecionar o nó para um ponto de destino diferente do original a uma determinada altura.

O utilizador necessita, então, de fornecer os pontos de partida e destino originais, bem como o ponto para o qual o nó deve ser redirecionado e o momento, em segundos, em que isso deve acontecer. Deve, também, especificar o nó para o qual esta mobilidade deve ser gerada.

A função efetua uma chamada à função `PerformRequest`, usando os pontos de partida e de destino originais, obtendo a lista de `Legs`, com todos os pontos da viagem original. Esta lista é depois percorrida e todos os pontos com tempo superior ao indicado pelo utilizador são removidos. É então efetuado uma segunda chamada à função `PerformRequest` usando, desta vez, como ponto de partida a latitude e longitude do último ponto na lista e como ponto de destino a localização especificada pelo utilizador. A lista recebida é adicionada à lista que contem os pontos anteriores ao tempo especificado pelo utilizador, que é, então, passada à função `SchedulePoints`, gerando a mobilidade propriamente dita.

`SchedulePoints(Ptr<Node>, std::list<Ptr<Leg> > legList)`

Esta função é essencial no funcionamento deste módulo, uma vez que é nesta classe que se efetua a criação de `Waypoints`, essenciais na geração de mobilidade.

A função percorre a lista de `Legs` recebida por parâmetro, retirando a lista de `Steps` contida em cada objeto do tipo `Leg`. Esta lista é, por sua vez, percorrida e é retirada a lista de `Points`

de cada `Step`. Esta lista é, também, percorrida e cada `Point` é convertido para um `Waypoint`, que é instanciado através do uso das coordenadas cartesianas e do tempo guardado em cada `Point`. Este `Waypoint` é adicionado ao `WaypointMobilityModel` através da chamada à função `AddWaypoint`, que adiciona o `Waypoint` criado a uma *double ended queue*.

No caso de ter ocorrido um erro e não ser adicionado nenhum `Waypoint` ao `WaypointMobilityModel`, é ativado um `NS_ASSERT_MSG` que termina a simulação fornece ao utilizador uma mensagem de erro explicativa.

7.1.3 Geração de mobilidade para um nó

Após a apresentação geral das classes e funções implementadas é necessário perceber o funcionamento do módulo, como um todo. Como tal, será apresentado nesta secção todos os passos de geração de mobilidade.

O utilizador necessita de, primeiro, instanciar um nó e, de seguida, atribuir-lhe o modelo de mobilidade `WaypointMobilityModel`. Após estes passos iniciais, o utilizador necessita de instanciar um objeto do tipo `RoutesMobilityHelper`, fornecendo ao seu construtor as coordenadas geográficas (latitude, longitude e altitude) do ponto real que pretende usar como centro do plano cartesiano. De seguida, o utilizador necessita de efetuar uma chamada

```
#include "ns3/core-module.h"
#include "ns3/mobility-module.h"
#include "ns3/netanim-module.h"
#include "ns3/routes-mobility-helper.h"
#include "ns3/config-store.h"
#include "ns3/trace-helper.h"
using namespace ns3;
NS_LOG_COMPONENT_DEFINE ("RoutesMobilityExample");
int
main (int argc, char** argv)
{
  double centerLat = 41.1073243, centerLng = -8.6192781, centerAltitude = 0;
  NodeContainer nodes;
  nodes.Create (1);
  MobilityHelper mobility;
  mobility.SetMobilityModel ("ns3::WaypointMobilityModel");
  mobility.Install (nodes);
  //É necessário instanciar o objeto RoutesMobilityHelper, passando-lhe as
  //coordenadas reais que deve usar como o centro do plano cartesiano.
  RoutesMobilityHelper routes (centerLat,centerLng,centerAltitude);
  //De seguida deve ser escolhida a rota desejada para o nó.
  routes.ChooseRoute("Instituto Superior de Engenharia do Porto, Porto, Portugal",
"Praça de Gomes Teixeira, Porto, Portugal",nodes.Get(0));
  Simulator::Run ();
  Simulator::Destroy ();
```

Figura 31 - Exemplo de código para realizar a geração de mobilidade para um nó

à função `ChooseRoute`, que se encarregará de gerar mobilidade para o nó. Um exemplo do código necessário, por parte do utilizador, para a geração de mobilidade pode ser visualizado na Figura 31.

Ao invocar a função `ChooseRoute`, o utilizador põe em marcha todos os componentes do módulo. Assim, esta função efetua uma chamada à função `PerformRequest` da classe `GoogleMapsApiConnect`, que trata o *input* do utilizador e submete o pedido à API.

Após a obter a resposta da API, é invocada a função `ParseXml`, que efetua uma analise ao XML, retirando a informação necessária à criação dos objetos `Leg` e `Step`. A classe `SaxHandler` recebe, no seu construtor uma referência para a variável de classe `m_legList`, que será preenchida com a informação contida na resposta da API.

De seguida, é invocada a função `ConvertToCartesian`, que recebe o atributo `m_legList` como argumento. Esta função começa por converter a informação contida no atributo `m_polyline` do objeto `Step` em `Points`, recorrendo à função `ConvertToGeoCoordinates`. A Figura 32 mostra o diagrama de atividades para esta função.

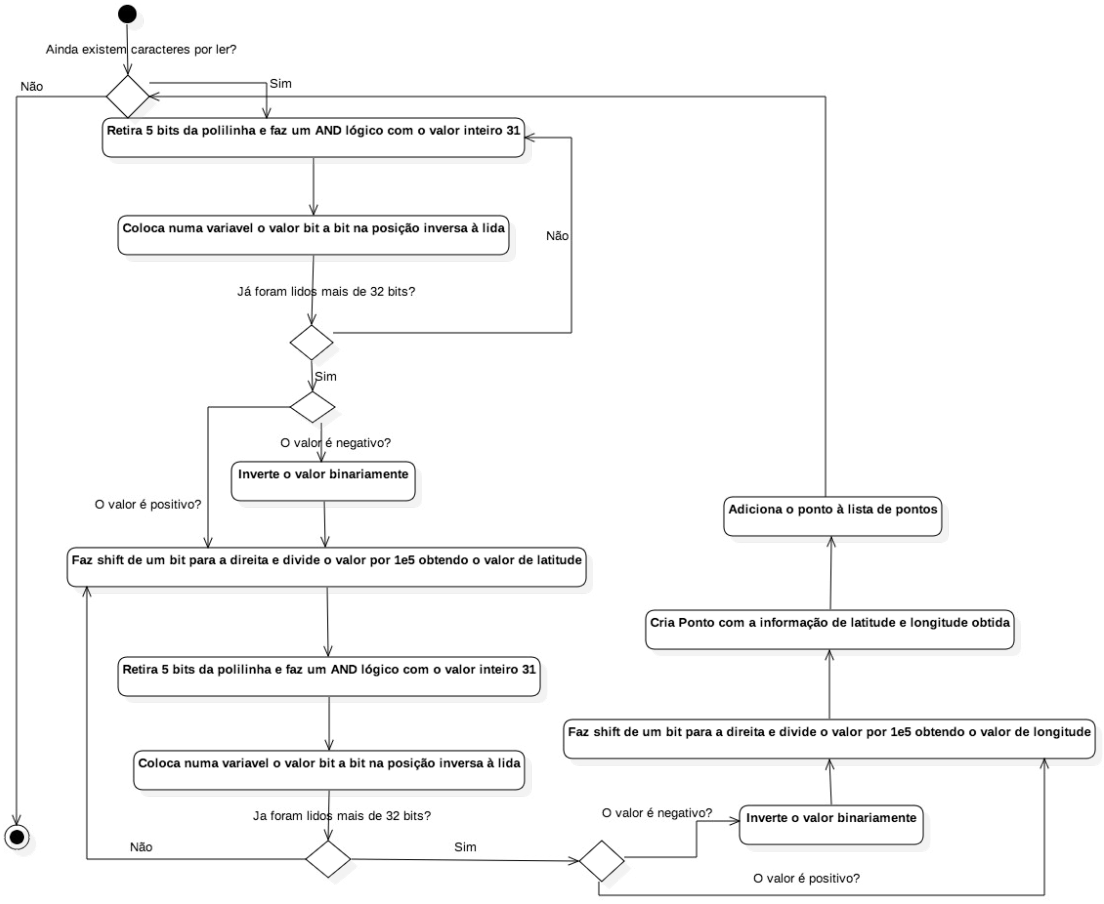


Figura 32 - Diagrama de atividades para a função ConvertToGeoCoordinates

A função converte, assim, os pontos contidos na polilinha de um `Step` e adiciona esses `Points` à lista de `Points` do `Step`. Após obter a informação geográfica dos `Points`, é então convertida essa informação para coordenadas cartesianas e, de seguida, é calculado o tempo a que o `Point` deve ser colocado.

A Figura 33 mostra um diagrama de atividades que ilustra o funcionamento da função `FillInWaypointTime`.

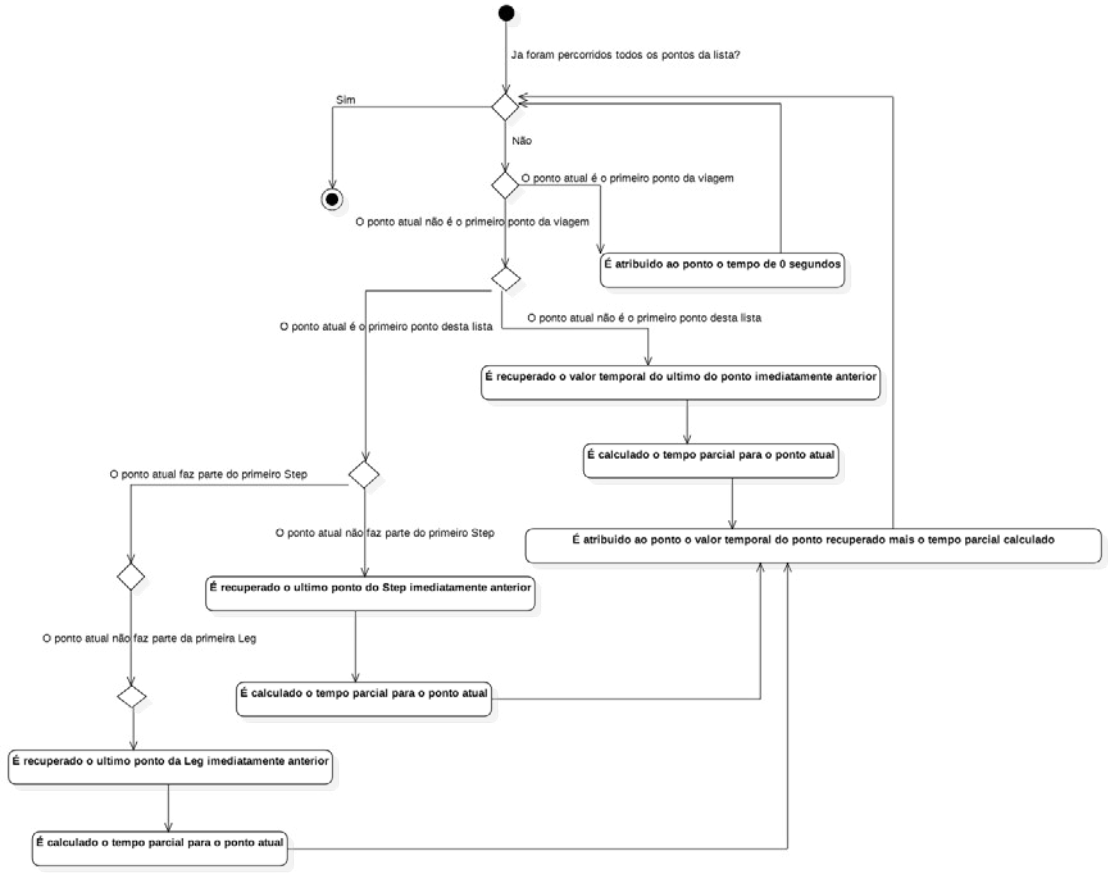


Figura 33 - Diagrama de atividades para a função FillInWaypointTime

Terminada esta função, o atributo `m_legList`, da classe `GoogleMapsApiConnect`, possui, então, a localização temporal e espacial de todos os `Points` da viagem. O atributo é retornado para a classe `RoutesMobilityHelper` onde, recorrendo à função `SchedulePoints`, os `Points` são adicionados ao `WaypointMobilityModel` de forma a gerar mobilidade.
A Figura 34 mostra o diagrama de sequência, ilustrando a troca de mensagens descritas neste capítulo do relatório.

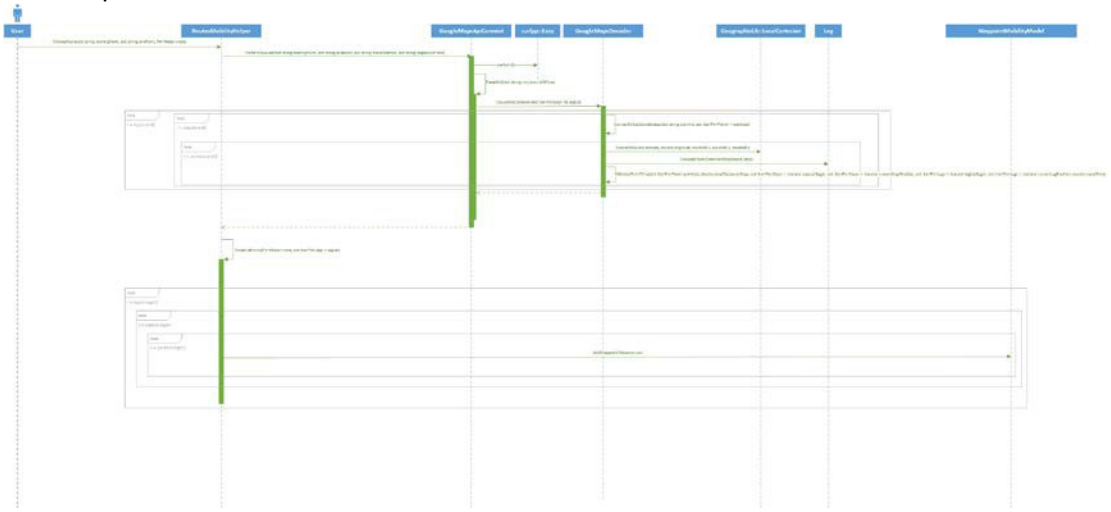


Figura 34 - Diagrama de sequência para o caso de uso "Geração de mobilidade para um nó"

7.1.4 Geração automática de mobilidade para um contentor de nós

Nesta secção serão apresentados os passos necessários para a geração automática de rotas de mobilidade para um contentor de nós, por parte do utilizador. Será também apresentado em detalhe o funcionamento interno do módulo na geração da mobilidade propriamente dita.

A atual implementação do módulo suporta a geração de mobilidade recorrendo a dois métodos distintos. É possível gerar mobilidade através da especificação de uma área, a partir de onde serão selecionados, aleatoriamente, dois pontos reais, que serão usados como ponto de partida e de chegada. É, ainda, possível gerar mobilidade através da obtenção de locais reais a partir da *Places API*. Esta API é capaz de retornar até 60 locais numa dada área de pesquisa, que serão posteriormente combinados para formarem os pontos de partida e de chegada dos nós.

Esta funcionalidade faz uso da funcionalidade discutida na secção 7.1.3 para a geração de mobilidade propriamente dita, focando-se, assim, na geração de locais, numa dada área, que possam ser usados pela função `ChooseRoute(std::string, std::string, Ptr<Node>)`.

Do ponto de vista do utilizador, os passos são semelhantes aos descritos na Figura 31, alterando apenas a função invocada. O utilizador começa por criar os nós e instala o modelo de mobilidade nos mesmos. Na Figura 35 é possível analisar os passos necessários para a geração automática de mobilidade para um dado contentor de nós.

```
#include "ns3/core-module.h"
#include "ns3/routes-mobility-helper.h"
#include "ns3/mobility-module.h"
#include "ns3/config-store.h"
#include "ns3/trace-helper.h"
using namespace ns3;
NS_LOG_COMPONENT_DEFINE ("RoutesMobilityExample");
int
main (int argc, char** argv)
{
  int numberOfNodes = 100;
  double centerLat = 41.1073243, centerLng = -8.6192781, centerAltitude = 0,
searchLat = 41.171770, searchLng = -8.611038, searchRadius = 5000;
  NodeContainer nodes;
  nodes.Create (numberOfNodes);
  MobilityHelper mobility;
  mobility.SetMobilityModel ("ns3::WaypointMobilityModel");
  mobility.Install (nodes);
  //É necessario, primeiro, instanciar o Routes Mobility Helper
  //Fornece-se ao construtor as coordenadas reais para serem usadas como centro do
plano cartesiano
  RoutesMobilityHelper routes (centerLat,centerLng,centerAltitude);
  //Esta função seleciona coordenadas aleatóriamente, como pontos de partida e de
chegada, contidos dentro da área especificada.
  routes.ChooseRoute (nodes, 41.161093, -8.633352, 41.146487, -8.606916);
  //Alternativamente, o utilizador pode, também, gerar mobilidade recorrendo a
coordenadas obtidas através da Places API.
  //A função recolhe locais reais contidos na área especificada pelo utilizador e
combina-os aleatóriamente como pontos de partida e de chegada
  //routes.ChooseRoute (nodes, searchLat, searchLng,searchRadius);
  Simulator::Run ();
  Simulator::Destroy ();
  return 0;
}
```

Figura 35 - Exemplo de código para realizar a geração de automática de mobilidade

Como ilustrado na Figura 35, o utilizador pode optar pela escolha de dois métodos para gerar mobilidade automaticamente. No código de exemplo apresentado encontra-se o método de geração de mobilidade através da escolha de coordenadas contidas numa área especificada e o método de geração de mobilidade através do recurso a locais obtidos por uma API externa, neste caso, a *Places API.*

No primeiro caso, a função `ChooseRoute` recebe os parâmetros máximos e mínimos de latitude e longitude, respetivamente. São, então, selecionadas aleatoriamente coordenadas que se encontram dentro da área especificada, recorrendo à função `GenerateRandomValue`, explicada anteriormente neste relatório.

Obtidas as coordenadas do ponto de partida e de chegada, estas são convertidas para *string*, no formato `latitude,longitude`, de forma a que possam ser usadas em conjunto com a função `ChooseRoute(std::string, std::string, Ptr<Node>)`, descrita em detalhe na secção 7.1.3. Estes passos são executados todos os nós presentes no

`NodeContainer`, desta forma gerando locais de partida e de chegada variados e distribuídos aleatoriamente pela área.

Como alternativa a este método, o utilizador pode indicar uma área a partir de onde serão, posteriormente, obtidos locais reais, recorrendo à *Places API*. Este método requer ao utilizador que especifique as coordenadas do centro geográfico da área onde devem ser pesquisados os locais, bem como o raio da pesquisa. A Figura 36 mostra um diagrama de sequência para esta função.

Recorrendo a este método, a função `ChooseRoute` começa por efetuar um pedido à *Places API*, recorrendo à função `PerformRequest` da classe `GoogleMapsPlacesApiConnect`. Esta função recebe as coordenadas do centro da pesquisa e o raio da mesma e é, assim, capaz de efetuar um pedido à *PlacesAPI* para locais contidos na área selecionada. Após receber a resposta da *Places API*, é invocada a função `ParseXml`, que irá percorrer o ficheiro XML retornado, criando objetos do tipo `Place`, que irá adicionar ao *vector* de `Places`, `m_placeList`. Tal como visto anteriormente, a *Places API* apenas retorna 20 locais por pedido e, como tal, é necessário repetir estes passos até que o atributo `m_nextPageToken` se encontre vazio, o que significa que não existem mais resultados disponíveis para aquele pedido.

Desta forma, são obtidos todos os locais disponíveis na área especificada e são, então, escolhidos para cada nó, os locais cujas coordenadas devem servir de ponto de partida e de chegada. Para tal, é invocada a função `GenerateRandomValue`, que gera dois valores entre 0 e o número máximo de `Places` contidos no *vector* de locais. Os valores obtidos são usados para obter o `Place` que se encontra na posição que o valor indica. Uma vez que os locais serão usados como pontos de partida e de chegada, os valores gerados são obrigatoriamente diferentes.

De seguida são convertidas para *string* as coordenadas geográficas dos dois locais, recorrendo à função `LocationToString`, e é invocada a função `ChooseRoute(std::string, std::string, Ptr<Node>)`, gerando, assim, mobilidade para um `NodeContainer` automaticamente.

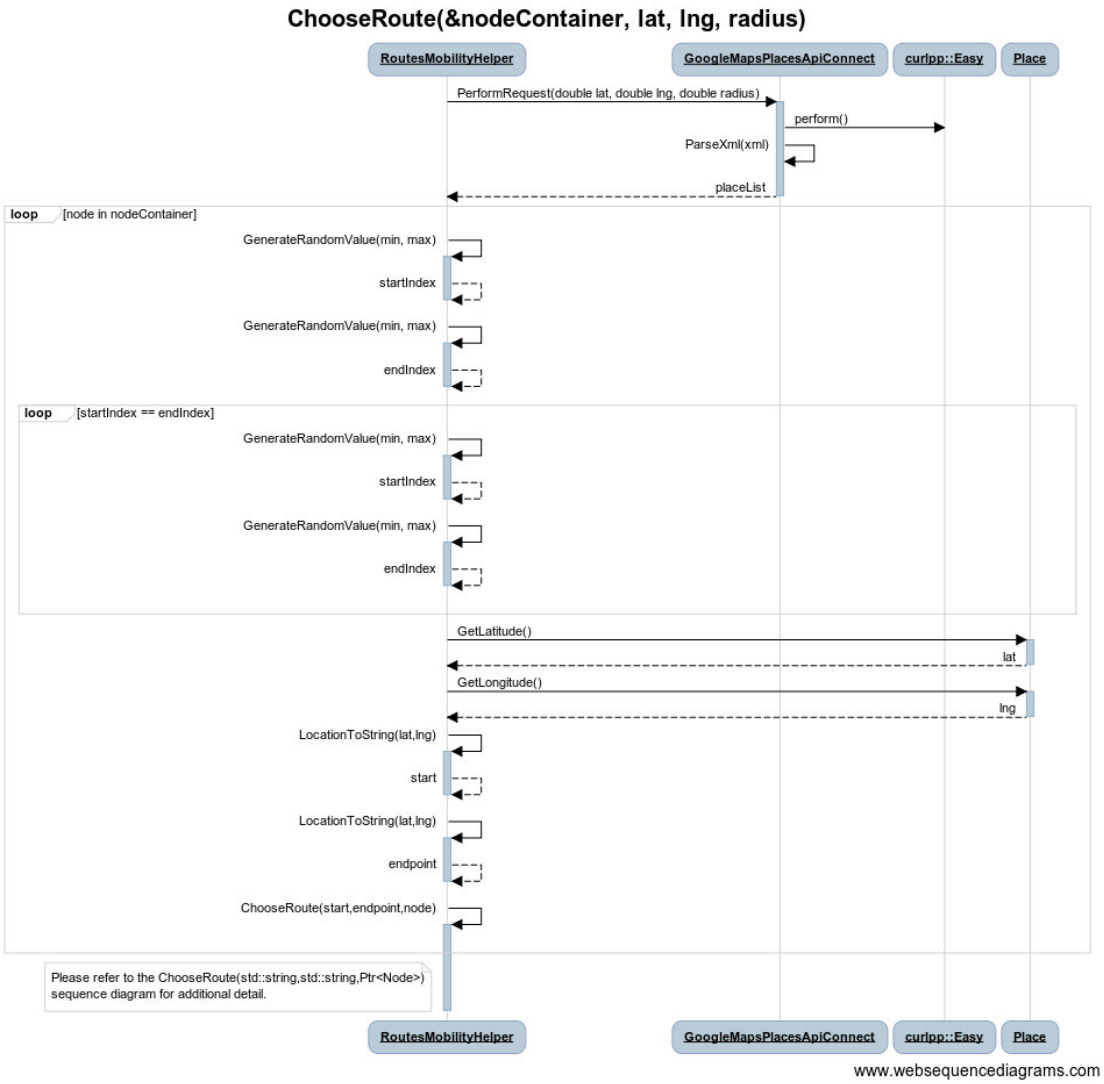


Figura 36 - Diagrama de sequência do método de geração de mobilidade automática recorrendo à Places API

7.1.5 Geração de mobilidade através de ficheiros XML locais

Nesta secção serão apresentados os passos necessários para a geração de mobilidade através do recurso a ficheiros locais. O módulo permite, ao utilizador, gerar mobilidade para nós sem a necessidade de possuir uma ligação à *internet*. Para o fazer basta especificar o caminho para o ficheiro XML ou para um diretório que contenha ficheiros XML, que possam ser usados com o módulo.

A Figura 37 mostra os passos que um utilizador deste módulo necessita de executar, com o objetivo de gerar mobilidade através do recurso a ficheiros XML previamente descarregados.

```
#include "ns3/core-module.h"
#include "ns3/mobility-module.h"
#include "ns3/config-store.h"
#include "ns3/netanim-module.h"
#include "ns3/routes-mobility-helper.h"
#include "ns3/trace-helper.h"
using namespace ns3;
NS_LOG_COMPONENT_DEFINE ("RoutesMobilityExample");
int
main (int argc, char** argv)
{
  int numberOfNodes = 3;
  double centerLat = 41.1073243, centerLng = -8.6192781, centerAltitude = 0;
  std::string path = "src/mobility-service-interface/examples";
  NodeContainer nodes;
  nodes.Create (numberOfNodes);
  MobilityHelper mobility;
  mobility.SetMobilityModel ("ns3::WaypointMobilityModel");
  mobility.Install (nodes);
  //É instanciado o RoutesMobilityHelper com as coordenadas para o centro do plano
  //cartesiano.
  RoutesMobilityHelper routes (centerLat,centerLng,centerAltitude);
  //É fornecido o contentor de nós para o qual deve ser gerada a mobilidade e o
  //caminho para a diretoria que contem os ficheiros XML.
  routes.ChooseRoute (nodes, path);
  //Alternativamente, o utilizador pode gerar mobilidade para um nó, indicando o
  //caminho para o ficheiro XML.
  //routes.ChooseRoute(nodes.Get(0),"/path/to/XML/file");
  Simulator::Run ();
  Simulator::Destroy ();
  return 0;
}
```

Figura 37 - Código de exemplo para o caso de uso "Geração de mobilidade através de ficheiros XML locais"

O utilizador necessita, primeiro, de criar os nós para os quais deve ser gerada a mobilidade, bem como proceder à instalação do modelo de mobilidade. De seguida, necessita de instanciar o objeto `RoutesMobilityHelper`, passando-lhe as coordenadas do centro do plano cartesiano, e invocar a função `ChooseRoute(NodeConteiner&, std::string)`. Ao invocar esta função, o utilizador dá instruções ao módulo para usar todos os ficheiros XML encontrados na pasta especificada, desta forma gerando mobilidade para o `NodeContainer` especificado.

A função `ChooseRoute` começa por abrir a diretoria especificada, retirando o nome de todos os ficheiros XML presentes. De seguida, a função executa, para todos os nós, a função `ChooseRoute(Ptr<Node>, std::string)`, usando o caminho completo dos ficheiros encontrados. Esta função, por sua vez, invoca a função `PerformOfflineRequest(std::string)`, da classe `GoogleMapsApiConnect`. De seguida é efetuada uma chamada à função `ParseXml`, que irá processar o ficheiro propriamente dito. Tal como explicado na secção 7.1.3, esta função irá ler o conteúdo XML,

retirando dele a informação necessária para a criação de objetos do tipo `Leg`, `Step` e `Point`. Da mesma forma, serão convertidas as polinhas em coordenadas geográficas e estas em coordenadas cartesianas, recorrendo à função `ConvertToCartesianCoordinates`, abordada em detalhe na secção 7.1.3. Será, ainda calculado o tempo a que cada `Point` deve ser colocado.

Por fim, com a informação necessária para a geração de mobilidade, é invocada a função `SchedulePoints`, abordada anteriormente neste relatório, que irá adicionar todos os `Points` criados ao `WaypointMobilityModel`.

7.1.6 Documentação

No decurso deste projeto foram elaboradas diferentes tipos de documentação detalhando diferentes aspetos da API do utilizador, bem como o funcionamento interno do módulo e as interações entre as suas classes.

O ns-3 faz uso da ferramenta Doxygen para documentar o código das suas classes e, como tal, foi usado no âmbito deste projeto para a documentação de todas as funções e classes introduzidas. São ainda documentados com esta ferramenta, os exemplos e o teste desenvolvido.

Foi ainda criado uma documentação geral do módulo, usando a ferramenta Sphinx, que oferece aos utilizadores e programadores um manual completo do funcionamento do módulo, abordando casos de uso avançados, resolução de problemas e arquitetura do módulo, entre outros assuntos.

Foi ainda elaborada uma página na *wiki* do ns-3 [47], a pedido de Tom Henderson, que documenta as várias fases de desenvolvimento do projeto, desde a sua fase inicial. É fornecida nesta *wiki*, também, a documentação UML deste projeto, que se encontra ainda no repositório de desenvolvimento do módulo e na documentação do mesmo.

7.1.7 Exemplos

A comunidade ns-3 encoraja os seus programadores e utilizadores a disponibilizarem pequenos pedaços de código que ilustrem um uso geral ou particular de um dado módulo. Como tal, o desenvolvimento deste módulo passou, também, pelo desenvolvimento de três exemplos, `routes-mobility-example.cc`, `routes-mobility-automatic-example.cc` e `routes-mobility-offline-example.cc`, que ilustram diferentes casos de uso deste módulo. O objetivo destes exemplos é o de fornecer ao utilizador, em

conjunto com a documentação existente, as bases para que este possa usar o módulo de acordo com as suas necessidades.

Os exemplos criados ilustram, na sua generalidade, o que foi abordado nas secções 7.1.3, 7.1.4 e 7.1.5.

7.1.8 Testes

A comunidade ns-3 encoraja também que os seus módulos possuam *suites* de testes unitários, de forma a manter a fiabilidade dos resultados. Como tal, foi desenvolvido no âmbito deste projeto, uma *suite* de teste para este módulo, que será apresentada nesta secção.

O teste confirma que os valores de latitude, de longitude e de tempo dos `Points` se mantem inalterados, desta forma garantindo que alterações futuras ao código não terão impacto no resultado final. A *suite* testa um total de 19 pontos distintos, de `Steps` distintos, obtidos através da leitura de um ficheiro XML. Os valores de referência são colocados manualmente, com os valores de latitude e longitude obtidos recorrendo à aplicação interativa de codificação de polilinhas[56], desenvolvida pela Google. Os valores das coordenadas cartesianas e de tempo foram calculados usando os algoritmos presentes no módulo para o efeito e atribuídos estaticamente. Desta forma é possível garantir que alterações ao código do módulo ou alterações da biblioteca GeographicLib não irão afetar o desempenho final do módulo. A Figura 38 ilustra um código simplificado do teste implementado no desenvolvimento deste projeto

```
void
RoutesMobilityModelTest::DoRun (void)
{
  Ptr<Leg> leg; Ptr<Step> step;
  std::list<Ptr<Leg> > legList; std::list<Ptr<Step> > stepList;
  std::list<Ptr<Point> > pointList, comparePointList, auxCmpPointList;
  std::list<Ptr<Step> >::iterator stepIt;
  legList = m_gmapsStrategy->PerformOfflineRequest ("src/mobility-service-inter-
face/test/routes-mobility-model-xml-test.xml");
  //Cria os valores expectaveis
  Ptr<Point> p1,p2,p3,p4,p5,p6,p7,p8,p9,p10,p11,p12,p13,p14,p15,p16,p17,p18,p19;
  p1 = new Point (41.18063,-8.609350000000001);
  (...)
  p19 = new Point (41.17640, -8.60389);
  p1->SetXCoordinate (1049.07); p1->SetYCoordinate (2053.2); p1->SetZCoordinate (-0.417406);
  (...)
  p19->SetXCoordinate (1507.3); p19->SetYCoordinate (1583.5); p19->SetZCoordinate (-0.37488);
  p1->SetWaypointTime (0.0);
  (...)
  p19->SetWaypointTime (141.0);
  pointList.push_back (p1);
  (...)
  pointList.push_back (p19);
  leg = (legList.front ());
  leg->GetStepList (stepList);
  step = stepList.front ();
  comparePointList = step->GetPointList ();
  stepIt = stepList.begin ();
  stepIt++;
  step = (*stepIt);
  auxCmpPointList = step->GetPointList ();
```

```
for(std::list<Ptr<Point> >::iterator i = auxCmpPointList.begin (); i!=auxCmpPoint-
List.end();i++)
    {
    comparePointList.push_back ((*i));
    }
  stepIt++; stepIt++;
  step = (*stepIt);
  auxCmpPointList = step->GetPointList ();
  for(std::list<Ptr<Point> >::iterator i = auxCmpPointList.begin (); i!=auxCmpPoint-
List.end();i++)
    {
    comparePointList.push_back ((*i));
    }
  for (std::list<Ptr<Point> >::iterator i = comparePointList.begin (),j = pointList.begin (); i
!= comparePointList.end (); i++,j++)
    {
      NS_TEST_ASSERT_MSG_EQ_TOL ((*i)->GetLatitude (), (*j)->GetLatitude (), 0.01, "The points
do not match!");
      NS_TEST_ASSERT_MSG_EQ_TOL ((*i)->GetLongitude (), (*j)->GetLongitude (), 0.01, "The
points do not match!");
      NS_TEST_ASSERT_MSG_EQ_TOL ((*i)->GetXCoordinate (), (*j)->GetXCoordinate (), 0.01, "The
points do not match!");
      NS_TEST_ASSERT_MSG_EQ_TOL ((*i)->GetYCoordinate (), (*j)->GetYCoordinate (), 0.01, "The
points do not match!");
      NS_TEST_ASSERT_MSG_EQ_TOL ((*i)->GetZCoordinate (), (*j)->GetZCoordinate (), 0.01, "The
points do not match!");
      NS_TEST_ASSERT_MSG_EQ_TOL ((*i)->GetWaypointTime (), (*j)->GetWaypointTime (), 0.01, "The
points do not match!");
    }
}
```

Figura 38 - Código simplificado do teste unitário desenvolvido

7.2 Aplicação CarCoDe

A aplicação CarCoDe, desenvolvida no âmbito deste projeto, implementa algoritmos de disseminação de dados em cenário de comunicação veicular. A aplicação tem como objetivo a disseminação de informações sobre diferentes serviços cujo interesse se limita à sua posição geográfica, como promoções restaurantes na área, bombas de gasolina, etc, numa rede veicular. A aplicação foi desenhada de forma a suportar veículos com rádio 802.11p e veículos com rádio LTE e com rádio 802.11p (híbrido), sendo assim capaz de suportar o modo de comunicação com a infraestrutura, através de LTE, e comunicação inter-veicular, através de 802.11p. A aplicação funciona através da comunicação, por parte dos veículos que possuem rádio LTE, com a infraestrutura central, descarregando informação sobre as *streams* a que estes veículos estão subscritos. A informação é, de seguida, disseminada para todos os veículos, recorrendo ao protocolo 802.11p.

Foram implementados dois algoritmos de disseminação distintos. Um algoritmo dissemina a informação epidemicamente, através de um mecanismo de *publish/subscribe* estendido através de um mecanismo de pedido/resposta (*request/response*). Neste cenário, os veículos comunicam as *streams* de dados que pretendem receber e os recetores deste pedido enviam a informação para a *stream* pedida, caso a possuam.

Na implementação do segundo algoritmo, foi implementado um mecanismo de *publish/subscribe*, com interações do tipo *push*. Usando este método, os veículos que

necessitam de informação para uma dada *stream* emitem pedidos regulares para essa informação. Os recetores destes pedidos, por sua vez, enviam a informação pedida, se a possuírem.

7.2.1 Levantamento de requisitos

O processo de desenvolvimento deste trabalho passou pelo levantamento de requisitos, em semelhança ao que já tinha sido feito para o módulo de mobilidade.

O trabalho realizado procurou cumprir ao máximo todos os requisitos definidos, que serão inumerados nesta secção.

7.2.1.1 Requisitos funcionais

Servidor:

Esta aplicação deve funcionar como um servidor externo à rede veicular, permitindo que os veículos com LTE comuniquem com um dado nó servidor, de forma a descarregar as informações para as *streams* a que se encontram subscritos.

Cliente:

Esta aplicação deverá funcionar nos nós veiculares da rede, permitindo que os veículos com LTE comuniquem com a infraestrutura e que todos os veículos sejam capaz de disseminar e interpretar as diferentes mensagens trocadas através do protocolo WSMP, abordado na secção 3.6.2 deste relatório.

Disseminação de dados:

A aplicação deve ser capaz de disseminar informação, recorrendo ao protocolo WSMP, para os veículos que se encontrem na sua área imediata. Deve, ainda, ser capaz de interpretar os diferentes tipos de mensagem e reagir de acordo com os mesmos.

Exportação de resultados:

Deve ser possível exportar os resultados obtidos no decurso das simulações para ficheiros de texto. Estes resultados devem refletir a comunicação efetuada entre os nós, sendo ainda necessária a exportação dos pacotes enviados e recebidos, ao nível da camada física e MAC.

Mobilidade:

Os nós simulados representam, na sua maioria, veículos, pelo que deve ser usado um modelo de mobilidade adequado, que confira aos nós uma mobilidade veicular realista. Os restantes nós devem encontrar-se estaticamente distribuídos pelo plano cartesiano.

***Script* de simulação:**

Uma vez que a aplicação gere, apenas, a comunicação entre os nós, deve ser criado um *script* de simulação capaz de montar a topologia de rede, atribuir os diferentes dispositivos de rede, instalar os diferentes modelos de mobilidade, instalar e configurar a aplicação, entre outras funcionalidades.

Parametrização da simulação:

A simulação deve ser parametrizável, devendo ser possível configurar diferentes aspetos da simulação a realizar a partir do ambiente de simulação

***Logging*:**

A simulação deve produzir dados, para posterior tratamento, que permitam avaliar o desempenho da rede. Deve, ainda, ser possível detetar a ocorrência de diferentes eventos de interesse.

7.2.1.2 Requisitos não funcionais

Durante o processo de desenvolvimento desta aplicação foram também identificados os requisitos não funcionais da mesma que, embora não estejam diretamente relacionados com as funcionalidades a implementar, restringem o desenvolvimento da aplicação.

De seguida serão apresentados os requisitos não funcionais identificados para esta aplicação.

Implementação:

A aplicação e o *script* de simulação devem ser desenvolvidos na linguagem C++, uma vez que é a linguagem utilizada no desenvolvimento dos diferentes componentes do simulador. Devem, ainda, ser respeitadas as normas de codificação do ns-3.

Usabilidade:

Deve ser simples de configurar, alterando o *script* de simulação, os diferentes parâmetros da aplicação e da topologia de rede em si.

7.2.2 Modelação e implementação

A aplicação descrita implementa a disseminação de dados, por parte dos veículos que possuem a informação necessária, para os veículos que não possuem informação, recorrendo, para isso, aos rádios 802.11p. Esta aplicação mistura a comunicação V2I com comunicação V2V, no sentido de que, os veículos que tem apenas rádio LTE, comunicam com a infraestrutura

para receber os dados, que serão depois disseminados para os restantes veículos da rede, por rádio 802.11p.

Para tal, foram criados objetos do tipo `Stream` e `StreamChunk`, que representam os dados que se pretende enviar. Uma `Stream` é definida como um conjunto de `StreamChunks`, tal como iremos ver numa próxima secção deste relatório. Foram, ainda, criadas as classes representativas da aplicação, `CarcodeClient` e `CarcodeServer`.

7.2.2.1 Classes implementadas

Tal como efetuado na secção anterior, foram analisadas as classes e módulos existentes no ns-3 de forma a perceber como se poderiam integrar as classes existentes com a aplicação a desenvolver.

Após esta análise, a escolha recaiu sobre o módulo *Applications*, uma vez que o objetivo central deste trabalho é o de criar aplicações que simulem o comportamento descrito, numa rede veicular. Para além disso, foi, também, imprescindível desenvolver classes representativas dos dados a transmitir, as classes `Stream` e `StreamChunk`. Uma *stream*, neste contexto, é um conjunto de informações, divididas em *chunks* e agrupadas por tema. As *streams* possuem, assim e entre outras coisas, um conjunto de *chunks*. Os *chunks* contêm pedaços de informação para uma *stream* em particular.

Foi, então, determinado que a arquitetura que melhor se enquadrava na solução do problema é a que pode ser visualizada na Figura 39.

A classe `CarcodeServer` (Figura 44) representa a aplicação responsável por manter a informação das *streams* disponíveis, globalmente. Esta aplicação responde, ainda, a pedidos de *streams* efetuados pelos veículos com capacidade de comunicação com a infraestrutura.

Por seu lado, a classe `CarcodeClient` (Figura 46), define o comportamento que deve ser adotado pelos nós veiculares da rede. Esta aplicação é responsável por efetuar a disseminação dos dados, de acordo com o algoritmo de pedido/resposta mencionado anteriormente. A aplicação é ainda responsável por gerir todo o processo relacionado com a troca de informação, desde o contato com o servidor externo, por parte dos nós com rádios LTE, à receção e processamento dos pacotes trocados através de rádios 802.11p. A classe `CarcodePushClient` (Figura 51) é, em tudo, semelhante à classe `CarcodeClient`, mas implementa um algoritmo de disseminação do tipo *publish/subscribe* com interações do tipo *push*.

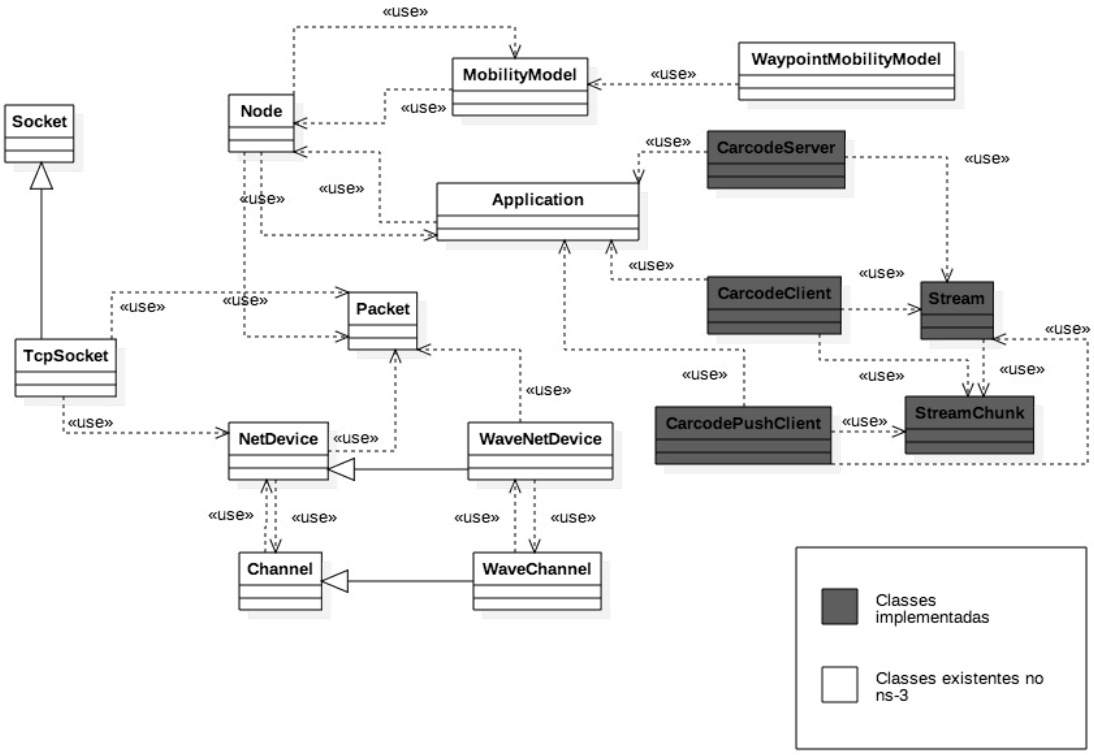


Figura 39 - Diagrama de classes simplificado para a solução desenvolvida

A classe `CarcodeServer` (Figura 44) representa a aplicação responsável por manter a informação das *streams* disponíveis, globalmente. Esta aplicação responde, ainda, a pedidos de *streams* efetuados pelos veículos com capacidade de comunicação com a infraestrutura.

Por seu lado, a classe `CarcodeClient` (Figura 46), define o comportamento que deve ser adotado pelos nós veiculares da rede. Esta aplicação é responsável por efetuar a disseminação dos dados, de acordo com o algoritmo de pedido/resposta mencionado anteriormente. A aplicação é ainda responsável por gerir todo o processo relacionado com a troca de informação, desde o contato com o servidor externo, por parte dos nós com rádios LTE, à receção e processamento dos pacotes trocados através de rádios 802.11p. A classe `CarcodePushClient` (Figura 51) é, em tudo, semelhante à classe `CarcodeClient`, mas implementa um algoritmo de disseminação do tipo *publish/subscribe* com interações do tipo *push*.

De seguida, serão abordadas individualmente, as classes mencionadas, detalhando os seus atributos e funções, bem como as suas principais responsabilidades.

`Stream`

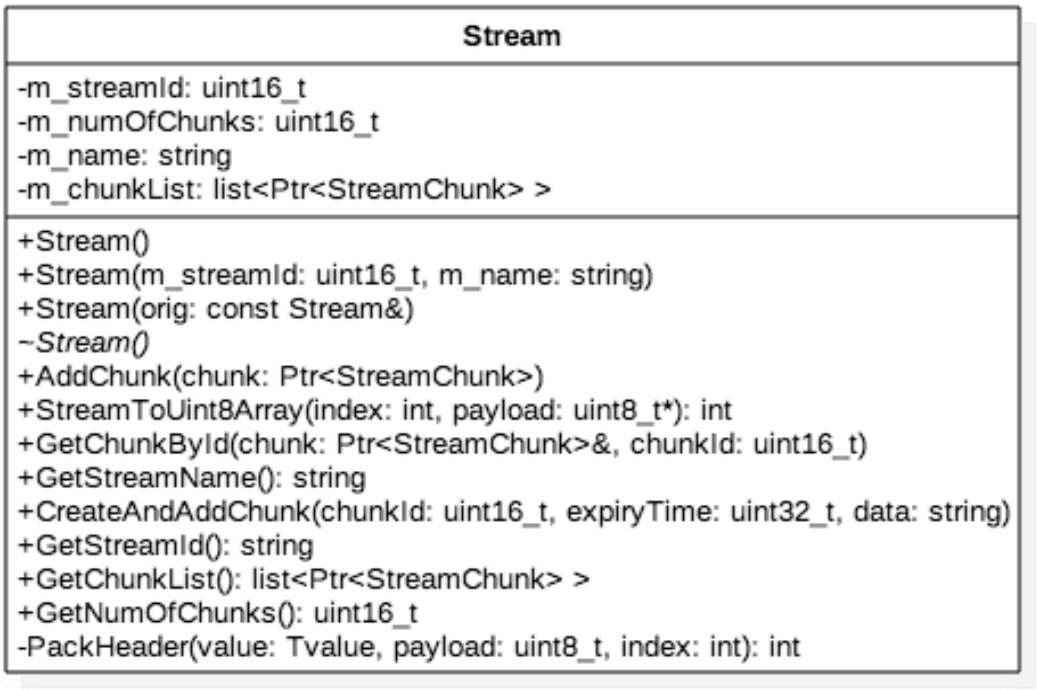


Figura 40 - Classe Stream

Esta classe representa uma *stream* de dados, responsável por agregar os *chunks* respetivos, a serem transmitidos durante a comunicação veicular. A classe possui, assim, uma lista de `StreamChunks` (`m_chunkList`), que contem os `StreamChunks` que aquela `Stream` possui. Existe, ainda, uma variável que contém código identificativo da `Stream` (`m_streamId`), o nome da `Stream` (`m_name`), e o número de `StreamChunks` que a `Stream` contem na lista.

A função `PackHeader(Tvalue value, uint8_t* payload, int index)` é uma função *template*, responsável por colocar um valor genérico (`value`), no vetor passado como parâmetro. De forma a lidar com tipos de dados superiores a 8 *bits*, a função recorre ao uso de *shifts* binários, de forma a colocar a informação no vetor, ordenada byte a byte. Esta função é crucial no funcionamento da função `StreamToUint8Array(int index, uint8_t* payload)`, que é responsável por colocar, no vetor fornecido, o código da `Stream`, bem como o número de `StreamChunks` presentes na lista. A Figura 41 mostra o formato desta serialização, bem como o tamanho dos respetivos atributos.

Figura 41 - Formato de serialização da classe Stream

A classe possui, ainda, um conjunto de funções utilitárias, que simplificar algumas operações. A função `AddChunk(Ptr<StreamChunk>& chunk)` e a função `CreateAndAddChunk(uint16_t chunkId, uint32_t expiryTime, std::string data)` permitem, respetivamente, adicionar um dado `StreamChunk` à lista de `StreamChunks` da classe e criar um novo `StreamChunk`, adicionando-o à lista de `StreamChunks`.

Esta classe possui, também, um conjunto de *getters* e *setters*, de forma a possibilitar o acesso aos seus atributos.

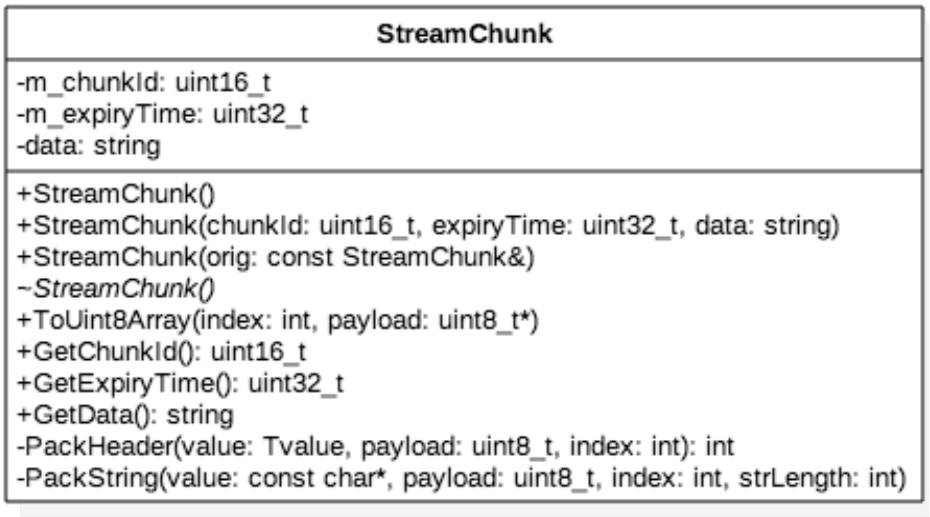


Figura 42 - Classe StreamChunk

`StreamChunk`

Esta classe representa um *chunk*, de uma dada *stream*. Possui, para isso, três atributos: um código identificativo do *chunk* (`m_chunkId`), o tempo em que o *chunk* se considera expirado, em EPOCH, (`m_expiryTime`) e os dados propriamente ditos (`m_data`).

As funções desta classe são responsáveis, principalmente, por serializar esta informação, de forma a poder ser enviada pela rede. Para tal, possui a função `PackHeader(Tvalue value, uint8_t* payload, int index)` que, em semelhança ao que foi descrito anteriormente, é responsável por colocar o valor de variáveis de diferentes tamanhos, no vetor passado a esta função. A função `PackString(const char* value, uint8_t* payload, int index, int strLength)` é semelhante à função `PackHeader`, na medida em que serializa os caracteres de uma dada *string*, no vetor fornecido.

A função `ToUint8Array (int index, uint8_t* payload)` faz uso das funções descritas no parágrafo anterior, efetivamente serializando o valor das variáveis desta classe no

vetor fornecido. A Figura 43 mostra o formato desta serialização, bem como o tamanho dos respetivos atributos.

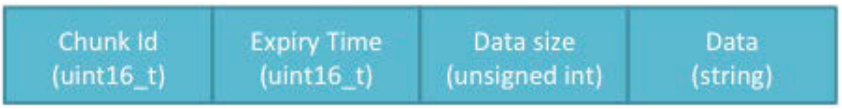


Figura 43 - Formato de serialização da classe StreamChunk

A classe possui, ainda, um conjunto de *getters* e *setters*, destinados a oferecer acesso às variáveis da mesma.

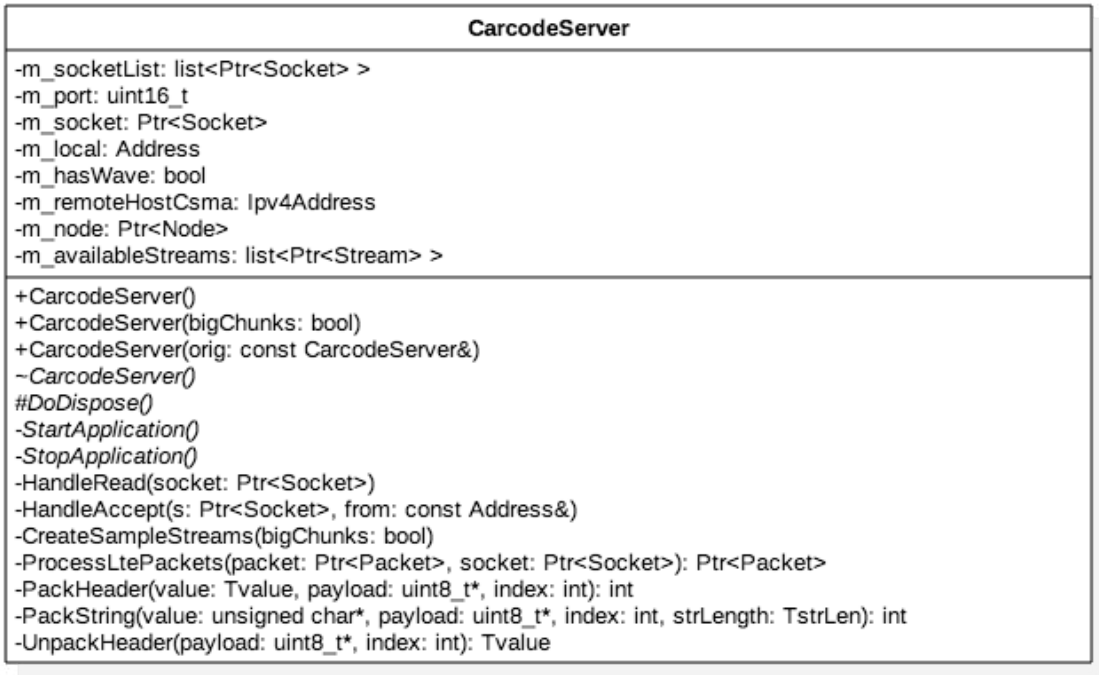


Figura 44 - Classe CarcodeServer

`CarcodeServer`

Esta classe representa uma aplicação, que modela o comportamento de um servidor de *streams*. Esta aplicação recebe e trata os pedidos de *streams*, enviados pelos veículos, por rádio LTE. Após o tratamento dos pedidos, a aplicação constrói e envia as *streams* pedidas ao veículo.

Como tal, esta aplicação possui uma série de funções dedicadas à receção e processamento dos pedidos. Possui, ainda, atributos que auxiliam na gestão da ligação à rede LTE e na gestão das diferentes *streams*. Assim, a função possui o número de porto do serviço (`m_port`), a *socket* que irá receber os pacotes provenientes da rede veicular (`m_socket`), o endereço local do nó (`m_local`)

A função `HandleRead (Ptr<Socket> socket)` é responsável por tratar os pedidos recebidos na interface point-to-point que se encontra ligada à rede LTE. A função é registada através recorrendo à função `SetRcvCallback`, da classe `Socket`, que recebe uma *callback* como argumento. Tal como descrito na secção 5.3, as *callbacks* são um método para associar uma dada função a um dado evento, neste caso à receção de um pacote.

Após a receção do pacote, a função `HandleRead` invoca a função `ProcessLtePackets(Ptr<Packet> packet, Ptr<Socket> socket)`, que irá processar o pacote propriamente dito, retirando a informação relativa à *stream*, através do uso da função `UnpackHeader`, que será abordada em mais detalhe num próximo parágrafo. De seguida, é também criado o pacote, que irá conter a informação relativa à *stream* pedida, incluindo todos os *chunks* conhecidos para aquela *stream*. Para tal, a função coloca no *vetor* o valor de controlo 0xFF00, que indica ao recetor que o pacote recebido é uma resposta a um pedido de *streams*, recorrendo à função `PackHeader`, que será explicada num próximo parágrafo deste relatório, e à função `StreamToUint8Array`, da classe `Stream`, que irá serializar no vetor a informação relativa à *stream* em questão. É, ainda, usada a função `ToUint8Array`, da classe `StreamChunk`, que é invocada em todos os *chunks* conhecidos para aquela *stream*, serializando-os no vetor. A Figura 45 ilustra o cabeçalho do pacote enviado.

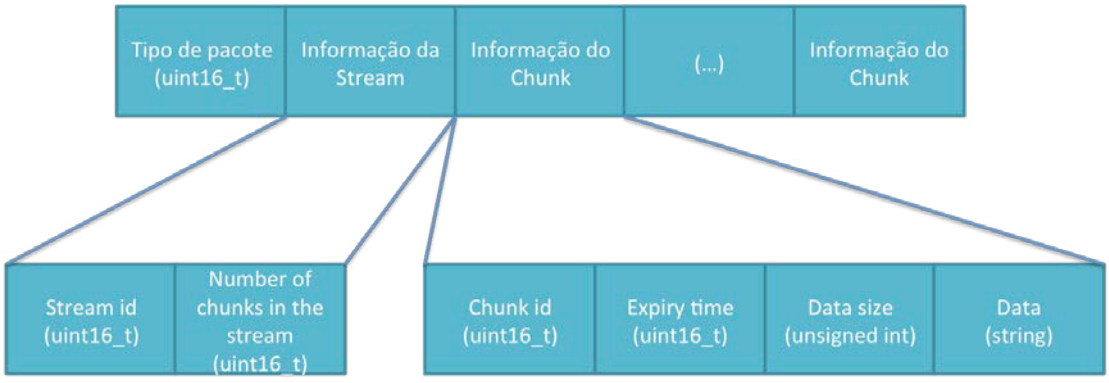


Figura 45 - Cabeçalho da resposta ao pedido de streams

Tal como mencionado anteriormente, esta classe possui, ainda, a função `PackHeader(Tvalue value, uint8_t* payload, int index)`, que é uma função *template,* responsável por colocar valores de tamanho variável no vetor fornecido, e a função `UnpackHeader(uint8_t* payload, int index)`, que acede a porções do vetor, de forma controlada, de forma a obter diferentes campos de informação. A função `UnpackHeader` é uma função *template*, que faz uso de *shifts* binários, de forma a retirar os *bits* necessários, pela ordem correta.

A função `HandleAccept(Ptr<Socket> s, const Address& from)`, por sua vez, é uma função necessária para o funcionamento do protocolo TCP, o protocolo escolhido para o transporte dos dados relativos a pedidos à infraestrutura, guardando numa lista os objetos do tipo `Ptr<Socket>`.

A classe possui, ainda, uma função utilitária, a função `CreateSampleStreams(bool bigChunks)`, que é responsável por criar um conjunto de diferentes *streams* de exemplo, bem como a informação dos respetivos *chunks*, a disponibilizar. A função recebe um booleano por argumento, que determina se o tamanho dos dados contidos nos *chunks* é normal, contendo apenas alguns bytes, ou grande, contendo até 1000 caracteres.

Esta classe implementa a interface `Application`, que define duas funções a implementar, a `StartApplication()` e a `StopApplication()`. Estas funções são funções especiais, que são invocadas de forma a dar inicio ou a terminar a aplicação em questão. A função `StartApplication()` é responsável por inicializar as *sockets*, bem como registar os *callbacks* necessários ao funcionamento da aplicação. A função StopApplication(), por sua vez, é responsável por realizar operações de limpeza, nomeadamente às *sockets* e às *callbacks* registadas, de forma a terminar a simulação efetivamente. O utilizador agenda estas funções para execução a partir do ambiente de simulação.

O construtor desta classe permite, ainda, ao utilizador, definir se os dados contidos nos *chunks* são de apenas alguns *bytes* ou se ocupam um *megabyte.* Este parâmetro deve ser manipulado a partir do ambiente de simulação.

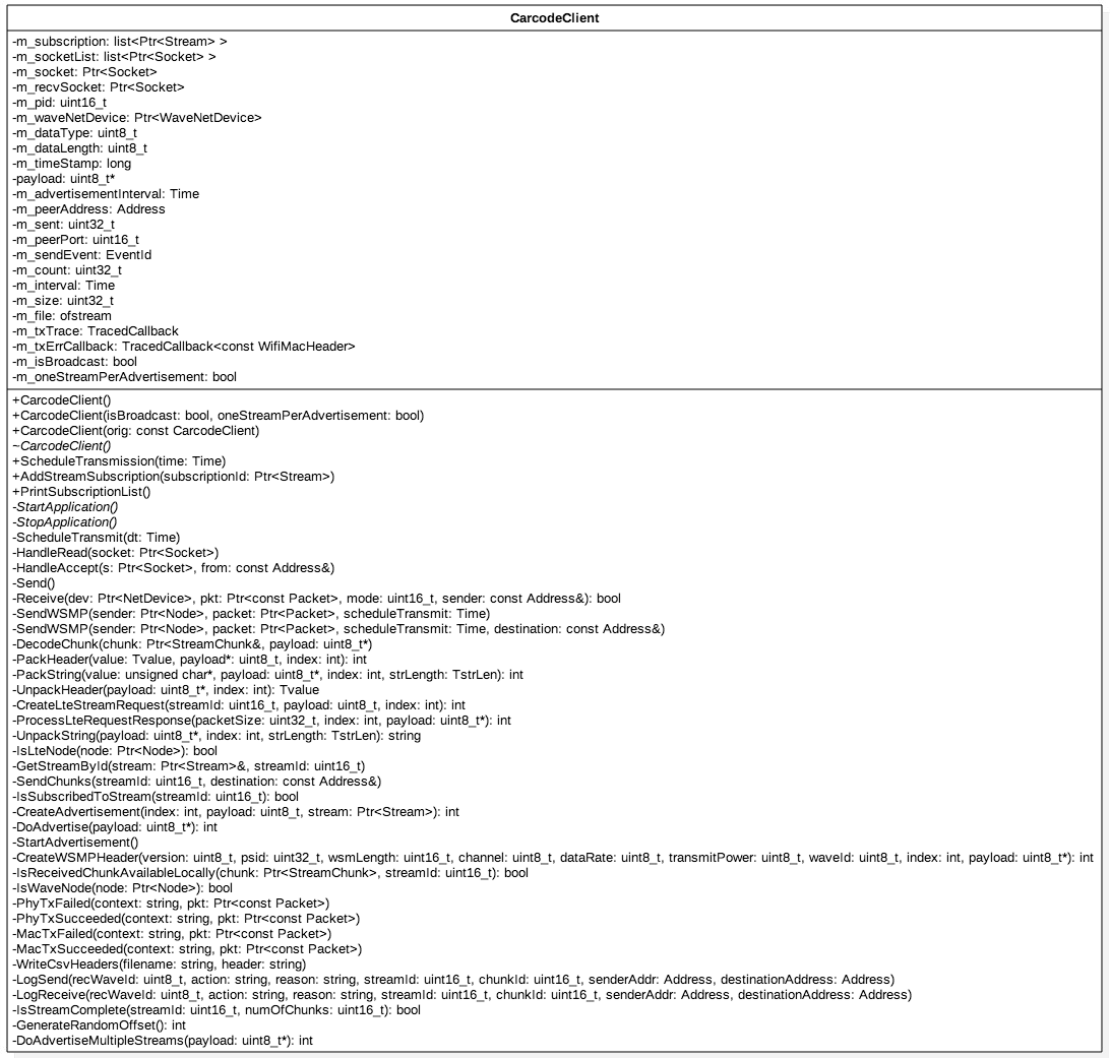


Figura 46 - Classe CarcodeClient

`CarcodeClient`

Esta classe é responsável por modelar o comportamento dos nós veiculares da rede, gerindo todo o processo de obtenção de *chunks* e posterior disseminação dos mesmos. Assim, a classe é responsável por permitir, aos nós com rádio LTE, contatar a infraestrutura de forma a obter as *streams* subscritas, gerindo ainda o envio de *advertisements*, que indicam, aos nós na área imediata, que existem nós interessados numa dada *stream*. A aplicação trata, ainda, do envio e processamento de *chunks*.

Esta classe possui, ainda, capacidades de *logging*, registando variados eventos de interesse, tais como eventos gerados pela própria aplicação (receção de *chunks*, envio de *advertisements*, etc) e eventos gerados por perdas de pacotes, entre outros.

A classe implementa, em semelhança à classe `CarcodeServer`, a interface `Application` e, assim, define as funções `StartApplication()` e `StopApplication()`. Estas

funções, em semelhança ao que já foi abordado na discussão da classe `CarcodeServer`, definem os comportamentos a adotar, de forma a inicializar e terminar corretamente a aplicação. A função `StartApplication` é, assim, responsável por inicializar as *sockets* e registar as *callbacks* de receção, envio e perda de pacotes, ao nível da camada física e camada MAC. É ainda responsável por inicializar os *advertisements*, através do recurso à função `StartAdvertisement()`, e por enviar o pedido inicial de *streams*, por parte dos nós com rádio LTE, recorrendo, para isso, à função `ScheduleTransmit(Time time)`.

A função `ScheduleTransmit` recebe por parâmetro o tempo a que deve ser agendado o evento de comunicação inicial com o servidor de *streams*. Com recurso à função `Simulator::Schedule`, é agendada a execução da função `Send()` para o tempo especificado na variável recebida por parâmetro.

A função `Send` é a função responsável por iniciar a comunicação com o servidor de *streams*, efetuando, assim, o pedido de informação para as *streams* a que o nó se encontra subscrito. Esta função recorre à interface LTE instalada em certos nós, para o contato com a infraestrutura. A função envia, por cada *stream*, um pacote contendo o pedido de *chunks* para uma dada *stream*. Esse pacote é construído com recurso à função `CreateLteStreamRequest(uint16_t streamId, uint8_t* payload, int index)`, que, por sua vez, recorre à função `PackHeader(Tvalue value, uint8_t* payload, int index)` para colocar o valor de controlo 0xFFFF, que indica ao servidor que o pacote é um pedido, bem como o código identificativo da *stream* pedida.

A Figura 47 mostra os diferentes campos do cabeçalho do pedido enviado.

Figura 47 - Cabeçalho do pedido de informação para uma stream

Durante a execução da função `StartApplication`, é registada, como *callback* de receção dos dispositivos LTE, a função `HandleRead (Ptr<Socket> socket)`. Esta função lida, diretamente, com as respostas do servidor de *streams*, descodificando as mesmas, através da função `ProcessLteResponse(uint32_t packetSize, int index, uint8_t* payload)`. Por sua vez, esta função retira da resposta enviada pelo servidor (Figura 45), a informação contida nos seus diferentes campos, criando os objetos `StreamChunk` respetivos e adicionando-os à lista de *chunks* da respetiva *stream*.

A função `StartAdvertisement` é responsável por dar início ao envio de *advertisements*, publicitando as *streams* a que um dado nó se encontra subscrito. Esta função recorre à função

`DoAdvertise(uint8_t* payload)` ou, em alternativa, à função `DoAdvertiseMultipleStreams(uint8_t* payload)`. Estas funções tem um comportamento similar, sendo que a primeira envia um *advertisement* por cada *stream* a que o nó se encontra subscrito, enquanto que a segunda envia apenas um *advertisement*, contendo todas as *streams* a que o nó se encontra subscrito. Este comportamento é controlado pela variável `m_oneStreamPerAdvertisement`. A função `StartAdvertisement` é executada periodicamente, recorrendo ao agendamento periódico da própria função.

A função `DoAdvertise` é a função responsável por construir um *advertisement* para uma dada *stream*. Esta função começa por colocar no pacote a enviar, o cabeçalho do protocolo WSMP, através da função `CreateWSMPHeader(uint8_t version, uint32_t psid, uint16_t wsmLength, uint8_t channel, uint8_t dataRate, uint8_t transmitPower, uint8_t waveId, int index, uint8_t* payload)`, que será explicada numa próxima secção deste relatório. De seguida, é invocada a função `CreateAdvertisement(int index, uint8_t* payload, Ptr<Stream>)`, que recorre à função `StreamToUint8Array`, já abordada, da classe Stream, efetivamente serializando a informação da *stream* no vetor. É, então, enviado o pacote, com recurso à função `SendWSMP(Ptr<Node> node, Ptr<Packet> pkt, Time t)`, que será abordada em maior detalhe numa próxima secção. O pacote contêm um campo de controlo (*WAVE element Id*), que indica o tipo de pacote. Este campo faz parte do cabeçalho WSMP e, por se tratar de um *advertisement*, toma o valor 0x82. O campo *Process ID* (PSID), do cabeçalho WAVE é, ainda, usado para guardar o código identificativo da *stream*. A Figura 48 permite visualizar os diferentes campos do cabeçalho de um *advertisement*, tal como é enviado pela função `DoAdvertise`.

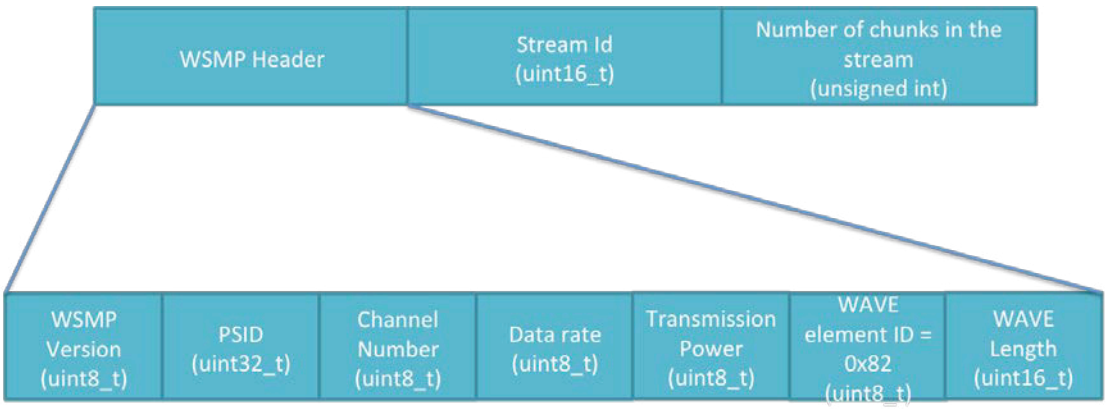


Figura 48 - Cabeçalho do pacote enviado pela função DoAdvertise

A função `DoAdvertiseMultipleStreams`, tal como referido anteriormente, tem um funcionamento muito similar ao da função `DoAdvertise`, enviando apenas um *advertisement*, que publicita todas as *streams* a que o nó se encontra subscrito.

A Figura 49 mostra o cabeçalho de um pacote enviado por esta função.

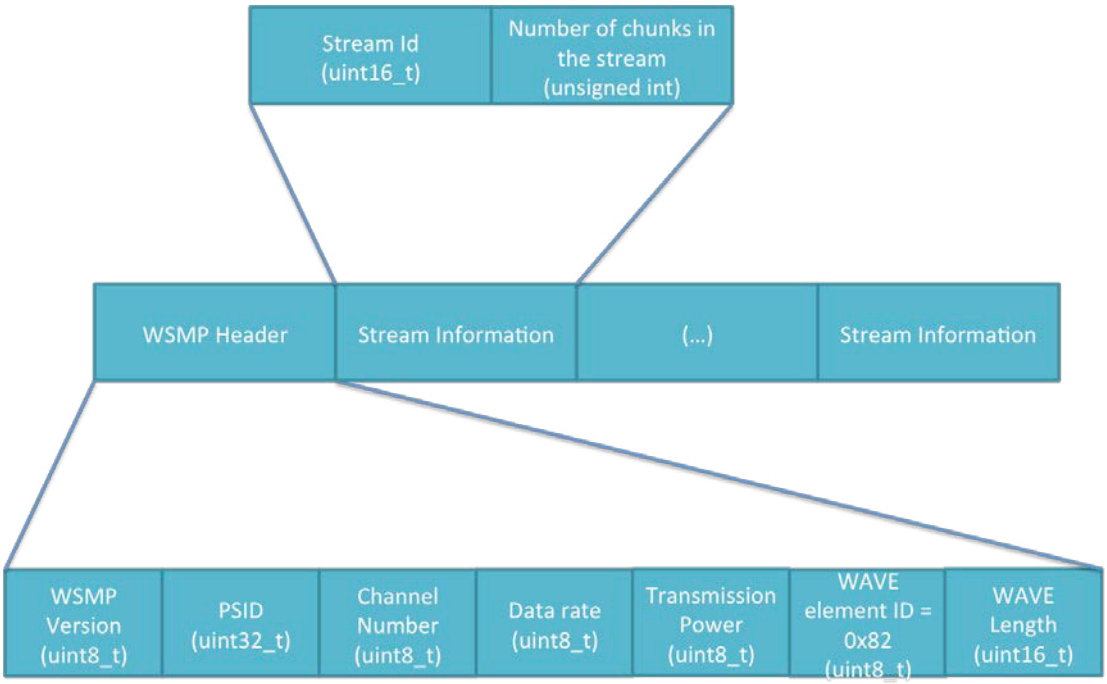


Figura 49 - Cabeçalho do pacote enviado pela função DoAdvertiseMultipleStreams

A função `Receive(Ptr<NetDevice> dev, Ptr<const Packet> pkt, uint16_t mode, const Address& sender)`, é crucial no funcionamento desta aplicação, uma vez que é a função responsável por tratar os pacotes recebidos após estes terem sido processados pelas camadas inferiores da pilha protocolar. A função começa por identificar o tipo de pacote recebido, retirando o valor do campo *WAVE element id*, do cabeçalho WAVE. Se o valor indicar que o pacote contêm um *advertisement*, é, então descodificada a informação relativa à *stream* recebida, nomeadamente o campo identificativo da mesma. É, então, iniciado o envio dos *chunks*, recorrendo à função `SendChunks(uint16_t service, Address& destination)`, que envia todos os *chunks* daquela *stream*, individualmente, para o endereço especificado. O cabeçalho do pacote enviado por esta função, contendo os dados de um *chunk* da *stream* especificada pode ser visualizado na Figura 50.

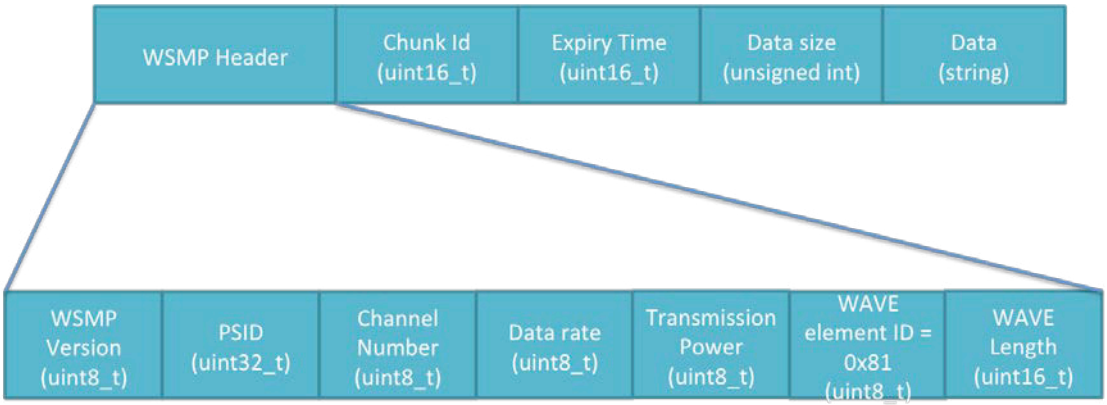


Figura 50 - Cabeçalho do pacote usado no envio de um chunk

A função `Receive` lida, ainda, com o processamento dos *chunks* recebidos. Como tal, é responsável por instanciar um objeto do tipo `Ptr<StreamChunk>`, usando a informação contida nos campos do pacote. Para isso, recorre à função `DecodeChunk(Ptr<StreamChunk>& chunk, uint8_t* payload)`, que se encarrega de retirar, do pacote, os campos referentes ao *chunk*, instanciando o objeto com a informação obtida. O *chunk* criado é, então, adicionado à lista de *chunks* da *stream* correspondente.

Existem, ainda, algumas funções que assistem na manipulação dos dados transmitidos, como a função `PackHeader(Tvalue value, uint8_t* payload, int index)`, `PackString(unsigned char* value, uint8_t* payload, int index, TstrLen strLength)`, `UnpackHeader(uint8_t* payload, int index)` e `UnpackString(uint8_t* payload, int index, TstrLen strLength)`. Estas funções, em semelhança ao que já foi explicado para outras funções similares, efetuam a serialização e desserialização, dos dados no vetor.

A classe possui, também, funcionalidades de *logging*, registando diferentes eventos da aplicação. As funções `PhyTxFailed(std::string context, Ptr<const Packet> pkt)`, `PhyTxSucceeded(std::string context, Ptr<const Packet> pkt)`, `MacTxFailed(std::string context, Ptr<const Packet> pkt)` e `MacTxSucceeded(std::string context, Ptr<const Packet> pkt)`, são registadas, na função `StartApplication`, como funções a serem executadas quando a comunicação tem sucesso, ou não, nas camadas física e MAC, respetivamente. As funções `LogSend(uint8_t recWaveId, std::string action, std::string reason, uint16_t streamId, uint16_t chunkId, Address senderAddr, Address destinationAddr)` e `LogReceive(uint8_t recWaveId,`

`std::string action, std::string reason, uint16_t streamId, uint16_t chunkId, Address senderAddr, Address destinationAddr)` registam eventos do tipo envio e receção, ao nível da camada aplicacional. Todas estas funções escrevem para diferentes ficheiros, na raiz da diretoria do ns-3, para mais tarde serem usadas na análise da respetiva simulação, como será visto num próximo capítulo deste relatório. Em adição a estes mecanismos de *logging*, é ainda registado o *output* da própria simulação, que contem alguma informação relevante.

Existe, ainda, um conjunto de funções utilitárias que assistem no funcionamento da aplicação, tais como as funções `AddStreamSubscription(Ptr<Stream> subscriptionId)`, responsável por adicionar a subscrição a uma dada *stream* a um nó, `GetStreamById(Ptr<Stream>& stream, uint16_t streamId)`, que retira a informação de uma dada *stream* da lista, retornando-a na variável recebida por parâmetro, e `GenerateRandomOffSet()`, que fornece um valor aleatório para o envio de *advertisements* para cada nó.

O construtor desta classe permite ao utilizador, a partir do ambiente de simulação, alterar o comportamento da aplicação. O utilizador pode especificar que pretende que a aplicação envie um pacote de *advertisement* por cada *stream* subscrita e que pretende que a transmissão dos *chunks* seja feita em *broadcast*.

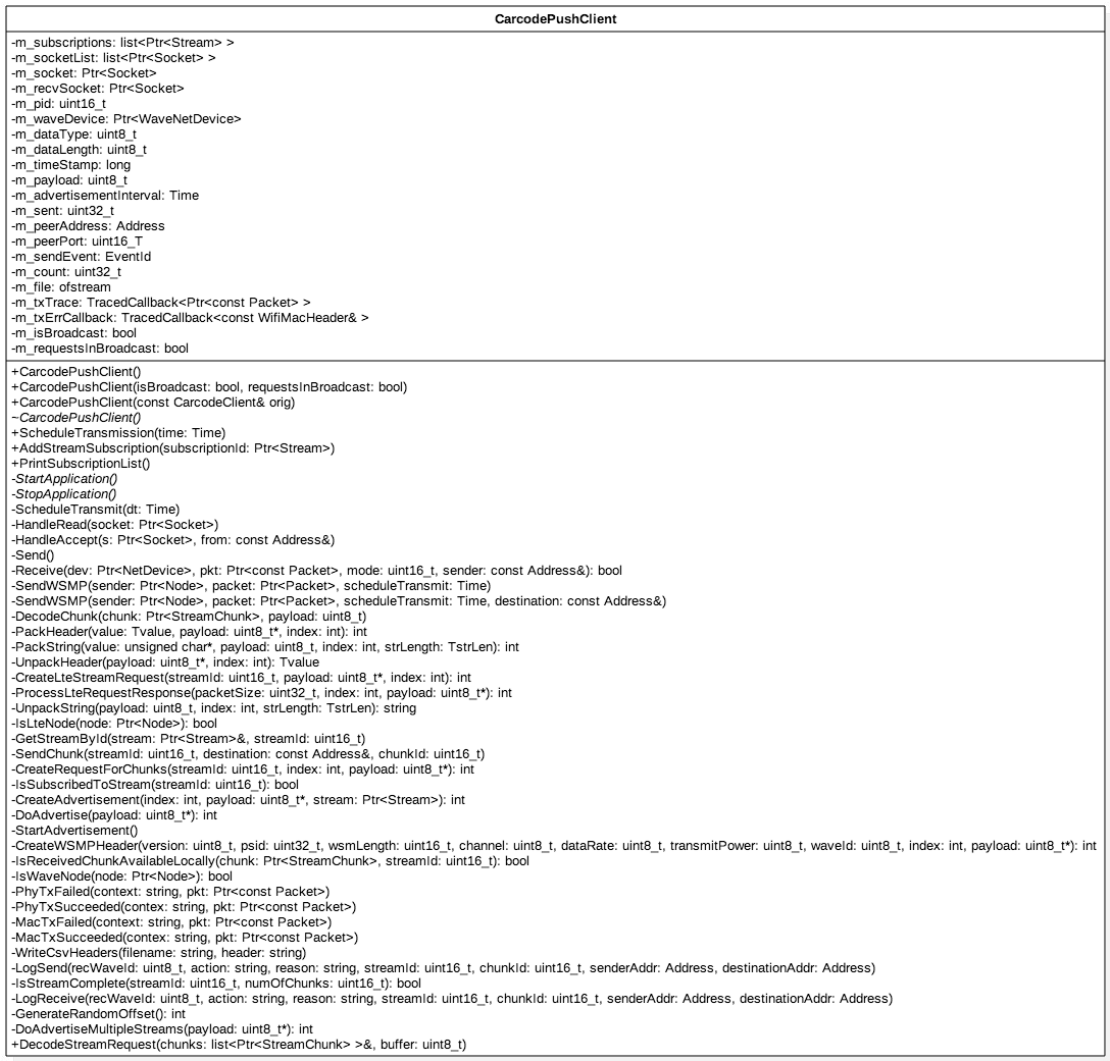


Figura 51 - Classe CarcodePushClient

`CarcodePushClient:`

Esta classe é similar à classe `CarcodeClient`, diferindo desta no algoritmo para a disseminação dos dados. A implementação, nesta classe, é feita através do algoritmo *publish/subscribe*, com interações *push*. Como tal, esta classe define a grande maioria das funções descritas na secção anterior, com pequenas modificações à função `Receive(Ptr<NetDevice> dev, Ptr<const Packet> pkt, uint16_t mode, const Address& sender)` e a adição das funções `SendChunk(uint16_t streamId, constAddress& destination, uint16_t chunkId)` e `DecodeStreamRequest(list<Ptr<StreamChunk> >& chunks, uint8_t* buffer)`, que serão abordadas de seguida.

Uma vez que esta aplicação modela um comportamento em que, de cada vez que é recebido um *advertisement*, o nó deve pedir a informação para aquela *stream*, foi necessário alterar o

comportamento da função `Receive` de forma a que, quando é recebido um *advertisement* para uma *stream*, seja criado um pedido de informações. Este pedido pode efetuado ao nó que emitiu o *advertisement* ou para todo o domínio de *broadcast*, que o utilizador pode definir a partir do ambiente de simulação. É, então, construído um pacote, recorrendo à função `CreateWSMPHeader`, que irá colocar o cabeçalho WAVE no pacote, e, de seguida, são verificados quais os *chunks* em falta, para a *stream* anunciada. É, então, criado um pacote pedindo todos os *chunks* em falta. Uma vez que o pacote é um pedido, o campo *Wave element Id*, do cabeçalho WAVE, contem o valor de controlo 0x80, que indica um pedido. O formato do pacote de pedido pode ser visto na Figura 52.

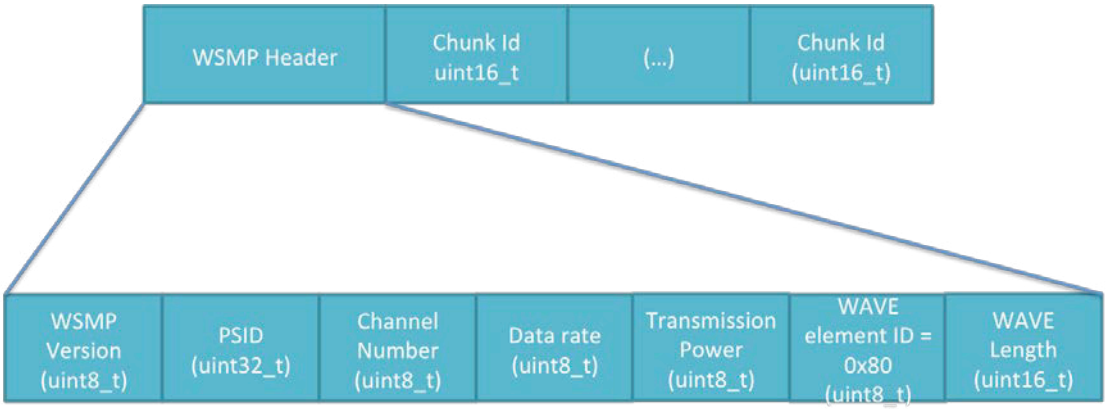


Figura 52 - Formato do pacote do pedido de chunks

Os pacotes deste tipo, ao serem recebidos, são primeiro interpretados, com recurso à função `DecodeStreamRequest`. Esta função é responsável por identificar os *chunks* pedidos, colocando-os numa lista. O nó irá, então, determinar se possui esses *chunks*, enviando-os, um a um, com recurso à função `SendChunk`. Esta função envia a informação que o nó possui para um dado *chunk*, para um outro nó, ou nós, que necessitem da informação.

O comportamento desta aplicação pode ainda ser alterado a partir do ambiente de simulação, à semelhança da classe descrita anteriormente, permitindo o envio de *chunks* em *unicast* ou *broadcast*, e permitindo o envio dos pedidos de informação em *unicast* ou *broadcast.*

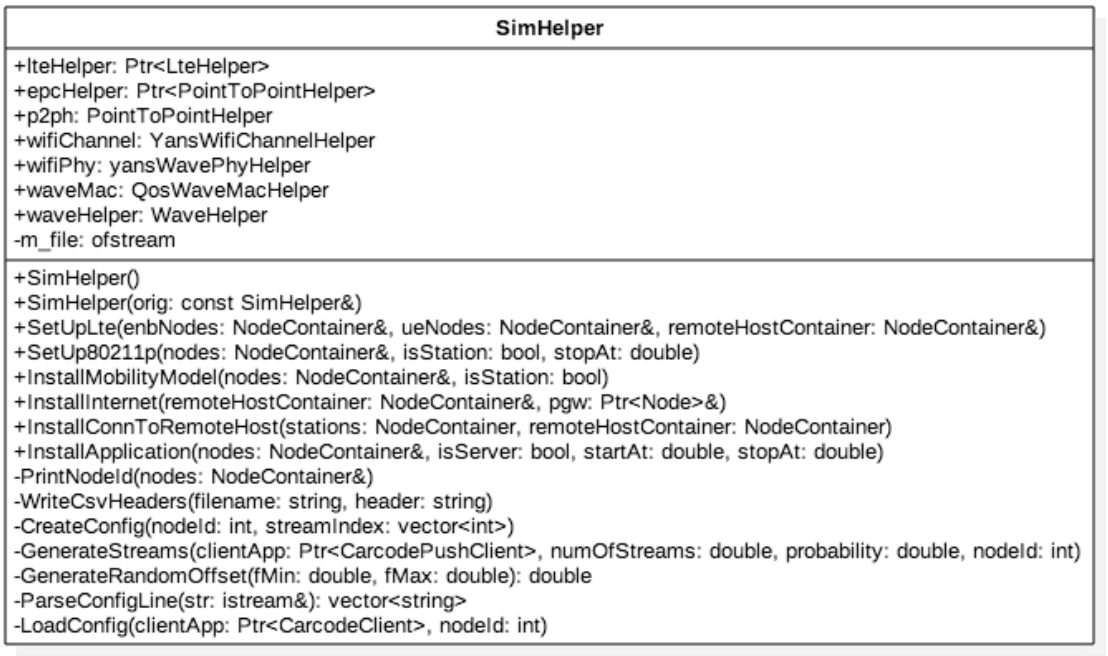


Figura 53 - Classe SimHelper

`SimHelper`

Esta classe, embora não faça parte do modelo aplicacional, é essencial ao funcionamento do módulo, uma vez que é responsável por realizar diversas operações com vista a preparar e iniciar a simulação propriamente dita.

Possui, para isso, um conjunto de funções que visam inicializar os nós com os respetivos dispositivos de comunicação, como é o caso das funções `SetUpLte(NodeContainer& enbNodes, NodeContainer& ueNodes, NodeContainer& remoteHostContainer)`, que instala nos nós os dispositivos LTE, e `SetUp80211p(NodeContainer& nodes, bool isStation, double stopAt)`, que instala os dispositivos WAVE nos nós. Esta função é ainda responsável por instalar a aplicação descrita neste capítulo, recorrendo à função `InstallApplication(NodeContainer& nodes, bool isServer, double startAt, double stopAt)` que, para além de instalar a aplicação correta para cada tipo de nó (cliente ou servidor), é ainda responsável por gerar, aleatoriamente, as *streams* a que cada veículo está subscrito ou por carregar, a partir de um ficheiro CSV, as configurações de *streams* de cada nó, o que permite realizar simulações sem alterações nas subscrições de *streams*.

A classe é ainda responsável por interligar as diferentes eNodeBs, através de uma ligação ponto a ponto, com o servidor central de *streams*, desta forma permitindo a comunicação *vehicle to infrastructure*, aos nós LTE.

7.2.2.2 Disseminação de dados através de algoritmo de pedido/resposta

Após a apresentação das classes da aplicação, é necessário fornecer uma visão geral do funcionamento da aplicação, bem como do algoritmo implementado.

Como já foi referido, este algoritmo funciona com o envio de *advertisements* das *streams* subscritas a intervalos regulares. Estes pacotes anunciam que o nó necessita de informação para aquela *stream*, ou *streams*, pelo que os recetores destes *advertisements* enviam, de imediato, todos os *chunks* que possuem para aquela *stream*.

Inicialmente, o utilizador necessita de definir o número de nós LTE e WAVE, bem como o tempo total de simulação. Necessita ainda de configurar os diferentes dispositivos, de forma a que possa existir comunicação entre os diferentes nós, bem como instalar a aplicação `CarcodeClient`. A classe `SimHelper` é responsável por assistir o utilizador nestas tarefas, fornecendo as funcionalidades necessárias à configuração de todos os aspetos da simulação.

Durante a execução da função `InstallApplication`, da classe `SimHelper`, são efetuados todos os passos necessários à configuração das aplicações a instalar. Assim, são geradas, as *streams* a que os nós devem estar subscritos, bem como selecionados os parâmetros da simulação. A atribuição destas subscrições é feita mediante a probabilidade de um nó subscrever a *stream*, fornecida pelo utilizador. É, ainda, invocada a função `ScheduleTransmission`, da classe `CarcodeClient`, que agenda um pedido, por parte dos nós que possuem rádios LTE, para contactar o servidor de *streams* externo, efetuando um pedido de informação para todas as *streams* a que estão subscritos. A Figura 54 mostra um diagrama de sequência simplificado que ilustra esta funcionalidade. O diagrama de sequência completo pode ser visualizado no Anexo 3.

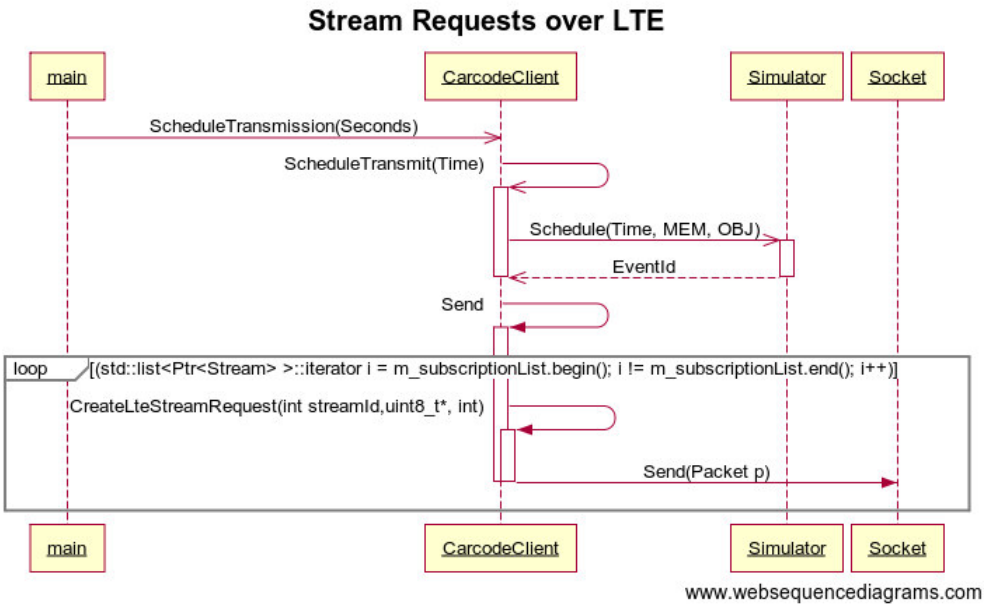


Figura 54 - Diagrama de sequencia simplificado para o pedido de streams por LTE

De seguida, são inicializadas as aplicações respetivas, `CarcodeServer` e `CarcodeClient`, recorrendo à função `StartApplication` da respetivas classe. Esta função é responsável por inicializar os dispositivos dos nós, efetuando todos os passos necessários para que o nó possa enviar e receber dados. É, ainda, agendado, para todos os nós, um envio de *advertisements* periodicamente, que serão enviados pela interface WAVE. Os *advertisements*, tal como vimos anteriormente neste relatório, publicitam as *streams* a que um dado nó está subscrito. A Figura 55 mostra um diagrama de sequência para esta funcionalidade, ilustrando as diferentes operações efetuadas na disseminação de *advertisements.*

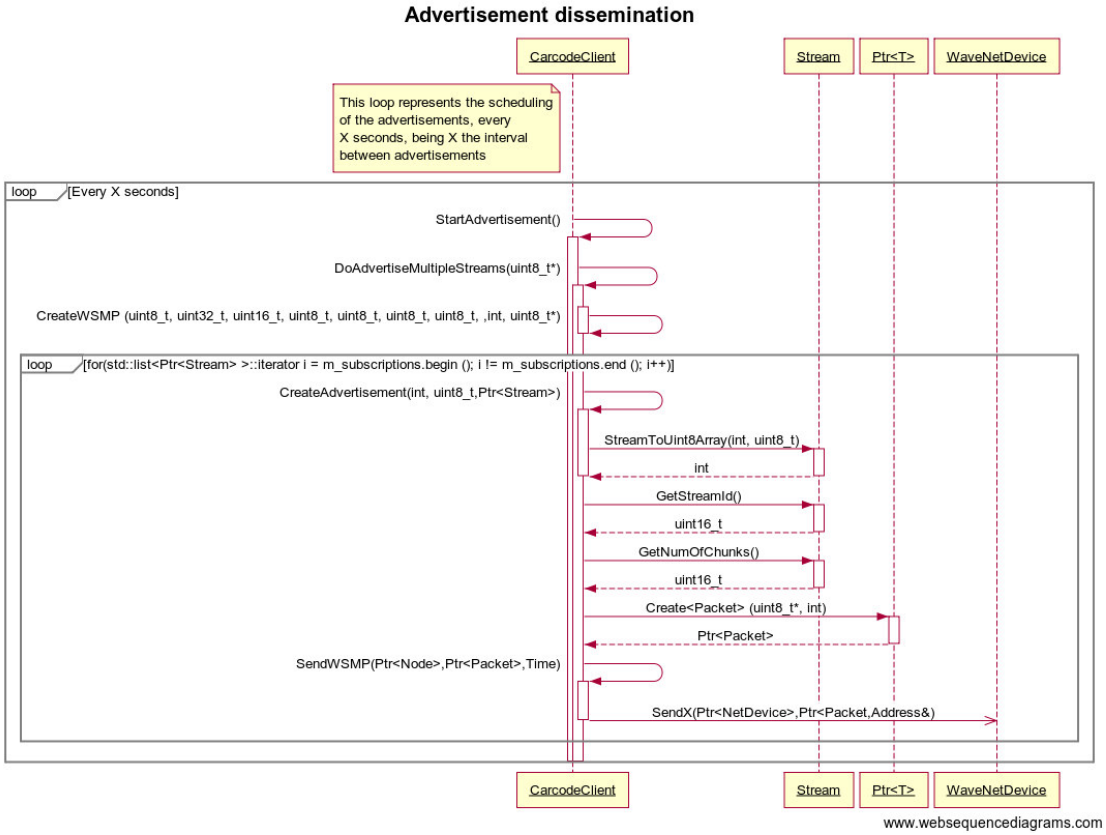


Figura 55 - Diagrama de sequencia da disseminação de advertisements

Quando um pacote WAVE é recebido, é invocada a função `Receive`, que irá decidir como manipular o pacote recebido. No caso dos pacotes contendo *advertisements,* são determinadas quais as *streams* publicitadas e enviados, recorrendo à função `SendChunks`, todos os *chunks*, das *streams* pedidas, que o nó possui. A Figura 56 ilustra a funcionalidade de envio de *chunks*, após a receção de um *advertisement*.

Da mesma forma, quando é recebido um pacote contendo um *chunk*, a função `Receive` adiciona-o à lista de *chunks* da respetiva *stream*, após verificar que não possui, localmente, o *chunk* recebido.

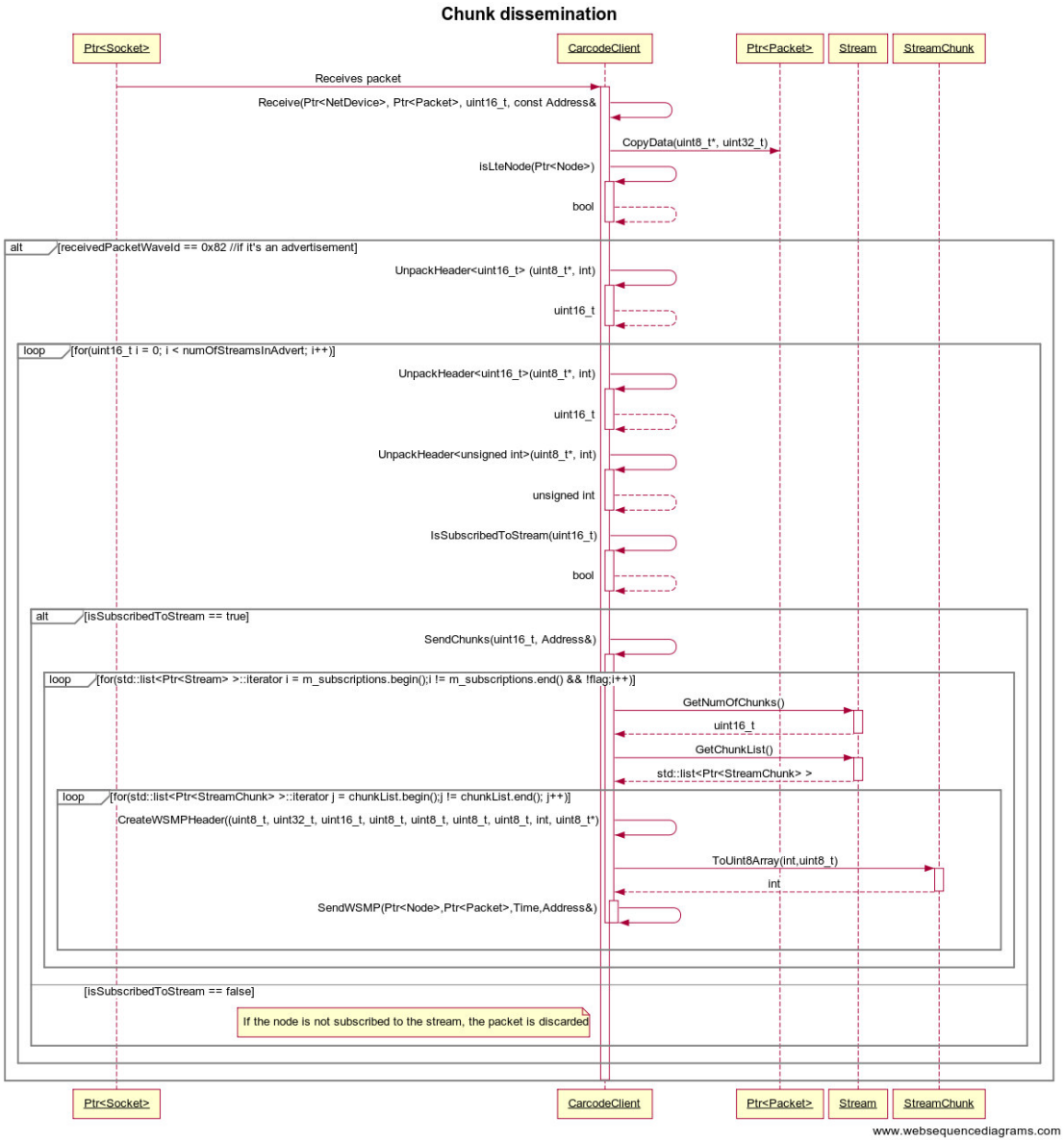


Figura 56 - Diagrama de sequencia para o envio de chunks

7.2.2.3 Disseminação de dados através de algoritmo do tipo *publish/subscribe*

Tal como referido anteriormente, esta aplicação permite, em adição ao algoritmo descrito anteriormente, o uso de um algoritmo de disseminação de informação do tipo *publish/subscribe* com interações *push*.

À semelhança do que foi descrito anteriormente, é necessário que o utilizador efetue as configurações necessárias, com vista a simular a rede pretendida. Para tal, o utilizador deve fazer uso da classe `SimHelper`, que fornece um conjunto de funções direcionadas para este fim. A função `InstallApplication` possui um papel fundamental nesta fase inicial, permitindo a parametrização da simulação a executar, bem como a geração aleatória das *streams* que os nós devem subscrever, à semelhança do que já foi descrito anteriormente.

Nesta função, são ainda instanciadas as duas aplicações, `CarcodeServer` e `CarcodePushClient`, que serão responsáveis pela comunicação efetuadas entre os nós.

Inicialmente, durante a execução da função `StartApplication`, da classe `CarcodePushClient`, são preparados os diferentes dispositivos de rede dos nós, de forma a que estes sejam capazes de lidar com o envio e receção de pacotes. Adicionalmente, os nós com rádio LTE efetuam um pedido de *streams*, a um servidor central. São, ainda, agendados o envio dos *advertisements* que, neste contexto, publicitam apenas a disponibilidade do veículo para responder a pedidos de informação para uma dada *stream*. Como tal, quando um

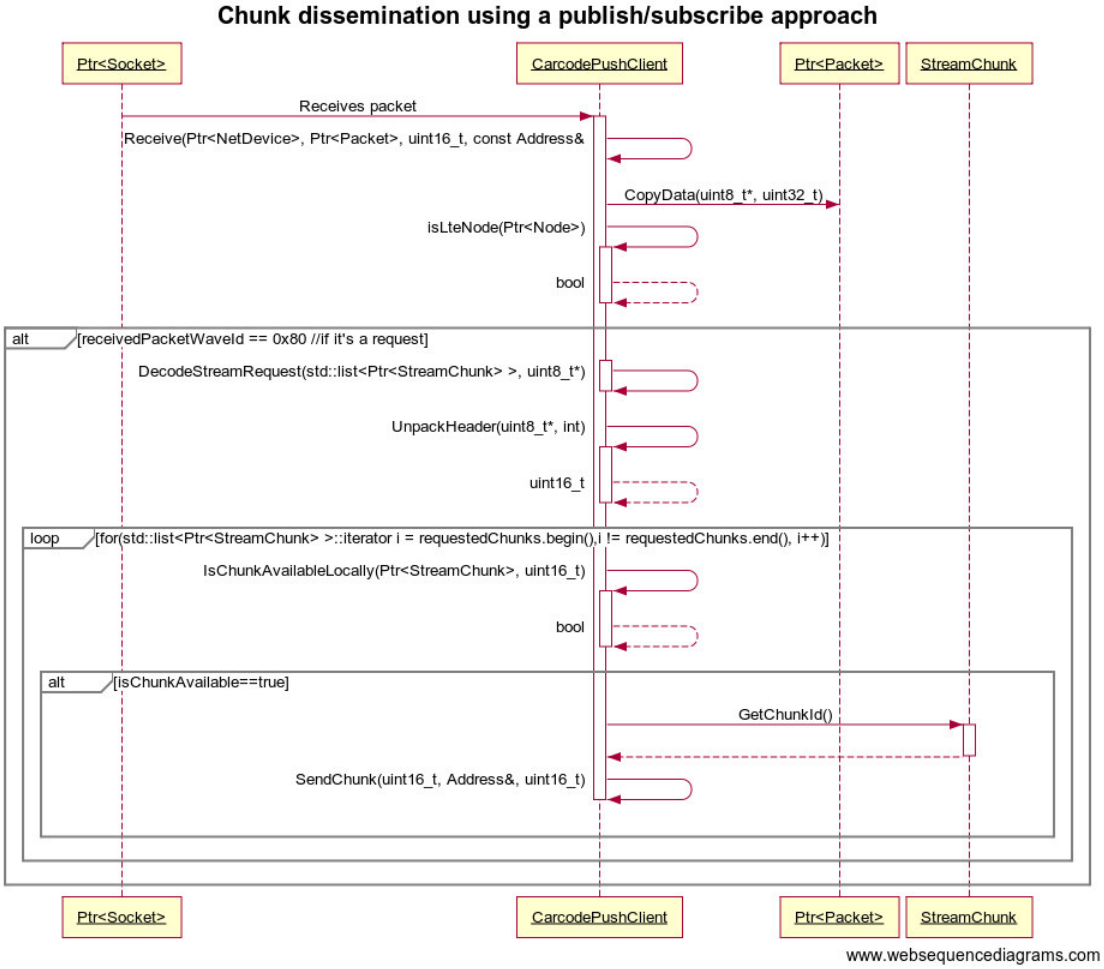


Figura 57 - Diagrama de sequencia da disseminação de chunks

advertisement é recebido pelo nó, é descodificada a informação relevante à *stream* publicitada e, se o nó se encontrar subscrito a essa *stream* e não possuir informação para a mesma, é efetuado um pedido de *chunks* pertencentes a essa *stream*. Por sua vez, quando um destes pedidos é recebido, o nó recetor prepara e envia, separadamente, todos os *chunks* que possui

para a *stream* pedida. A Figura 57 mostra um diagrama de sequência, que ilustra a receção de um pedido de *chunks*.

Quando um *chunk* é recebido, o mesmo é descodificado e, se o nó ainda não possuir esse *chunk*, este é adicionado à lista de *chunks* da *stream* correspondente.

7.2.2.4 Exportação de resultados

As aplicações descritas neste capítulo possuem a capacidade de exportar os resultados da simulação, registando, assim, eventos considerados importantes para o estudo. Para tal, foram invocadas *callbacks* para registar o envio, receção e rejeição de pacotes, tanto ao nível da camada física como da camada MAC. Adicionalmente, é ainda efetuado o registo de certos eventos da própria aplicação. A exportação dos resultados, propriamente dita, é feita através da escrita para diferentes ficheiros CSV.

As *callbacks* são registadas com a invocação da função `TraceConnect`, nos objetos representativos da camada física e MAC. As *callbacks*, tal como abordado anteriormente neste relatório, registam uma função para ser executada quando um determinado evento é detetado. Assim, foram registadas um total de 10 *callbacks*, `MacRx`, `MacTx`, `MacRxDrop`, `MacTxDrop`, `PhyRxDrop`, `PhyTxDrop`, `PhyRxBegin`, `PhyTxBegin`, `PhyRxEnd` e `PhyTxEnd`. As *callbacks* `MacRx` e `MacTx` registam os eventos de receção e envio de cada pacote, respetivamente, na camada MAC. Por sua vez, as *callbacks* `PhyTxBegin`, `PhyTxEnd`, `PhyRxBegin` e `PhyRxEnd`, registam o envio e receção dos pacotes, respetivamente, na camada física. As *callbacks* `PhyTxBegin` e `PhyRxBegin` registam quando um pacote começa a ser transmitido pelo dispositivo, enquanto que as *callbacks* `PhyTxEnd` e `PhyRxEnd` registam uma transmissão completa do pacote. De forma a detetar os pacotes perdidos, são registadas as *callbacks* `PhyTxDrop`, `PhyRxDrop`, `MacTxDrop` e `MacRxDrop`. Estas callbacks registam a perda de pacotes durante a transmissão e receção do mesmo, ao nível das camadas física e MAC, respetivamente. O registo dos eventos é efetuado para um ficheiro CSV único para cada *callback*.

A função `PhyTxSucceed` é registada para as *callbacks* `PhyTxEnd`, `PhyRxEnd`, `PhyRxBegin` e `PhyRxEnd`, que regista o envio ou receção de dados, na camada física. Esta função regista, para o ficheiro CSV da *callback*, o código identificativo do nó, o tipo de pacote, o código identificativo da *stream* e do *chunk*, o endereço MAC do transmissor, o endereço MAC de destino e o tempo a que o evento ocorreu. Da mesma forma, a função `MacTxSucceeded` é registada para as *callbacks* `MacTx` e `MacRx`, registando o envio e receção de pacotes, respetivamente, ao nível da camada MAC. Os eventos de envio e receção são registados para

o ficheiro CSV da *callback* respetiva, assim como o código identificativo do nó, o tipo de pacote, o código identificativo da *stream* e do *chunk* e o tempo a que o evento ocorreu.

De forma a registar as perdas de pacotes na camada física, é registada a função `PhyTxFailed`, para as *callbacks* `PhyRxDrop` e `PhyTxDrop`. Esta função regista, de cada vez que a transmissão ou receção de um pacote falha, o código identificativo do nó, o tipo de pacote, o código identificativo da *stream* e do *chunk*, o endereço MAC do transmissor, o endereço MAC de destino e o tempo a que o evento ocorreu. Da mesma forma, para registar os pacotes perdidos ao nível da camada MAC, é registada a função `MacTxFailed` para as *callbacks* `MacTxDrop` e `MacRxDrop`. Esta função regista, de cada vez que um pacote falhou a transmissão ou receção ao nível da camada MAC, o código identificativo do nó, o tipo de pacote e o tempo a que o evento ocorreu.

Existem, ainda, as funções `LogSend` e `LogReceive`, que são usadas para registar eventos considerados interessantes durante a execução da aplicação. Estas funções geram informação, para um ficheiro CSV com o mesmo nome, com o seguinte formato: código identificativo do nó, tipo de pacote, a ação tomada, o motivo que levou à criação do registo, o código identificativo da *stream* e do *chunk*, o endereço MAC do transmissor, o endereço MAC de destino e o tempo a que o evento ocorreu. Desta forma, as funções são usadas para registar eventos como a receção ou o envio de um *chunk*, a receção ou envio de *advertisements* e registar que um nó terminou a receção de todos os *chunks* existentes para uma dada *stream*, entrou outros.

Adicionalmente, existem também instruções `NS_LOG`, que imprimem diversas informações relativas ao funcionamento da aplicação. Estas informações são registadas para ficheiro, através do redireccionamento do *output*, durante a execução da simulação.

7.2.2.5 Mobilidade dos nós

Uma vez que as simulações executadas pretendiam modelar uma rede veicular, foi necessário conferir mobilidade aos nós que representam veículos. Para este efeito, foi usado o `RoutesMobilityModel`, descrito na secção 7.1 deste relatório. Assim, foi possível modelar diferentes cenários de trânsito, em diferentes cidades, de forma a conferir um maior grau de realismo ao estudo. Para as simulações efetuadas foram usados métodos de geração automática de mobilidade, nomeadamente através do recurso à *Places api*.

8 Resultados e validação

Este capítulo irá apresentar os resultados de simulações realizadas no ns-3, no âmbito deste projeto, com o objetivo de validar o módulo de mobilidade implementado e avaliar o desempenho de algoritmos de disseminação de dados.

8.1 RoutesMobilityModel

De forma a validar o módulo de mobilidade foram efetuadas simulações, baseadas no *script* de simulação `vanet-routing-compare.cc`, disponível como parte do módulo WAVE. Os resultados descritos nesta secção foram apresentados na conferência *Workshop on ns-3* 2015 (WNS3), a principal conferência do consórcio ns-3 [32].

O *script* `vanet-routing-compare.cc` oferece aos seus utilizadores a possibilidade de comparar o desempenho de diferentes protocolos de *routing*, estando atualmente implementados os protocolos Ad hoc On Demand Distance Vector (AODV), Destination-Sequenced Distance Vector (DSDV), Optimized Link State Routing Protocol (OLSR) e Dynamic Source Routing (DSR). O *script* dispõe de alguns cenários pré-definidos, aos quai foram adicionados, para o estudo, três novos cenários. Os cenários adicionados diferem no modelo de mobilidade usado. No primeiro cenário foi usado o `RoutesMobilityModel`, no segundo foi usada mobilidade gerada através do simulador SUMO, e no terceiro foi usado o `RandomWaypointMobilityModel`.

As simulações pretendem medir:

- O realismo da mobilidade gerada, através da analise visual das trajetórias.
- O tempo necessário para configurar a simulação
- O desempenho do protocolo de *routing* na comunicação, uma vez que este é afetado pelo modelo de mobilidade usado[57].

Os cenários adicionados consistem em simulações de 99 nós, durante 300 segundos simulados. Os nós enviam mensagens de segurança 10 vezes por segundo, a 6Mbs por segundo e o modelo de propagação usado foi o *TwoRayGround*.

A mobilidade é o fator variável entre estas simulações, tendo sido selecionados 3 modelos de mobilidade diferentes que geraram trajetórias numa área aproximadamente igual. No cenário de SUMO, a mobilidade foi gerada numa área de 4.6 quilómetros de largura e 3.0 quilómetros de altura no centro da cidade de Barcelona, enquanto que no cenário de `RoutesMobiliyModel`, foi selecionada uma área circular de 2.1 quilómetros de raio.

Nestes dois cenários, a área escolhida para a geração de mobilidade foi aproximadamente na mesma zona da cidade. No cenário do `RandomWaypointMobilityModel` foi usada uma área de 4.6 quilómetros de largura por 3.0 quilómetros de altura.

As simulações foram efetuadas com o objetivo de validar a hipótese da existência de elevadas similaridades entre a mobilidade obtida com recurso a um simulador de trânsito complexo e realístico, como o SUMO, e a mobilidade obtida com recurso ao `RoutesMobilityModel`. Da mesma forma, pretendia-se verificar que, em comparação com o `RandomWaypointMobilityModel`, existissem poucas, ou nenhumas, similaridades, uma vez que este modelo de mobilidade é um modelo simples, que não tem em conta a rede de estradas em que os veículos se movem.

Os resultados desta simulação avaliaram o *overhead* de comunicação, no rácio entre o total de bits enviados pelo rádio 802.11p e o total de bits correspondentes à camada física e MAC. A Figura 58 mostra que o desempenho do `RoutesMobilityModel` é similar ao observado no cenário de SUMO, particularmente depois de serem ultrapassados os instantes iniciais [58], convergindo ao longo do tempo com o cenário SUMO. O cenário do `RandomWaypointMobilityModel` mostra uma profunda diferença para com os outros dois cenários.

Assim, podemos afirmar que o modelo de mobilidade descrito na secção 7.1 deste relatório permitiu que o protocolo AODV tivesse um desempenho semelhante ao observado no cenário de SUMO, enquanto se destaca, pela positiva, comparavelmente ao `RandomWaypointMobilityModel`. Os resultados reportados são os valores médios obtidos através da execução de 10 simulações distintas.

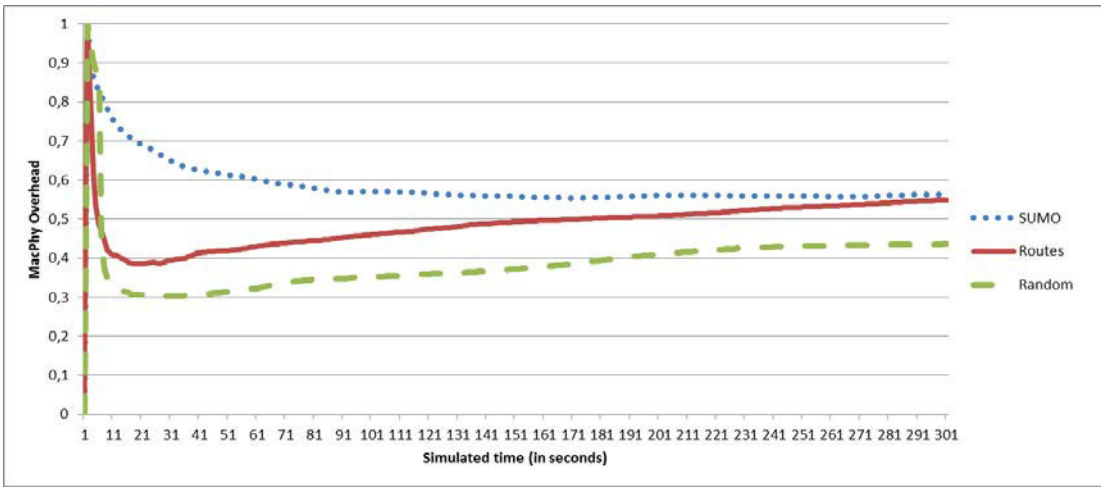


Figura 58 - Overhead do protocolo AODV na comunicação

O cenário de SUMO demorou algumas horas a configurar, devido à complexidade dos diferentes *scripts* necessários à geração da mobilidade, contudo tanto o cenário do `RoutesMobilityModel` como do `RandomWaypointMobilityModel` demoraram apenas alguns minutos, e foram efetuadas com relativa simplicidade.

A Figura 59, Figura 60, Figura 61 e Figura 62 mostram os cenários simulados, com os nós e

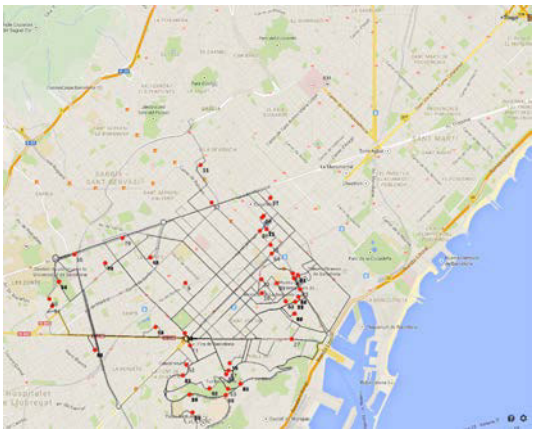


Figura 59 - Cenário RoutesMobilityModel

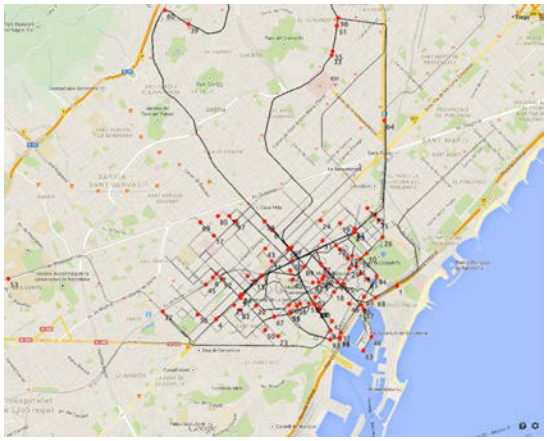


Figura 60 - Cenário SUMO

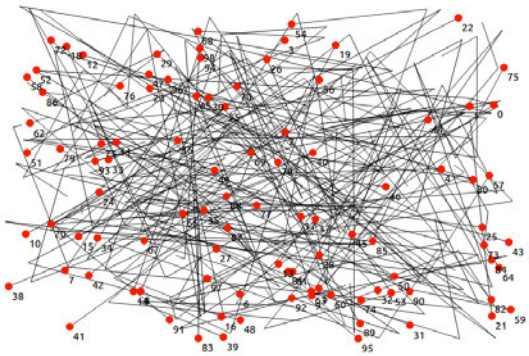


Figura 61 - Cenário RandomWaypointMobilityModel

Figura 62 - Simulação de mobilidade de 1000 nós

respetivas rotas sobrepostos a um mapa real da área. Tal como é possível observar, no cenário de SUMO e no cenário de `RoutesMobilityModel`, o percurso dos nós adere à rede de estradas existente, sendo bastante semelhante entre os dois cenários. Contudo, no cenário de `RandomWaypointMobilityModel`, os percursos dos nós não tem em conta a rede de estradas na área considerada e é muito diferente dos outros dois cenários em estudo.

A Figura 62 mostra uma simulação diferente, de 1000 nós dispostos aleatoriamente numa área selecionada pelo utilizador. Os nós escolheram, aleatoriamente, pontos de partida e de destino contidos na área fornecida. O objetivo desta simulação foi o de comprovar a viabilidade deste modelo de mobilidade com simulações contendo elevados números de nós, mantendo a mesma usabilidade e complexidade de configuração do `RandomWaypointMobilityModel`, e um elevado realismo de simulação.

8.2 Aplicação CarCoDe

A análise à aplicação CarCoDe foi efetuada com recurso a um *script* de simulação desenvolvido especialmente para o efeito, responsável por configurar os dispositivos de rede dos nós, instalar e configurar as aplicações e aplicar mobilidade aos nós. Este capítulo irá mostrar os resultados obtidos para as simulações com o algoritmo de pedido/resposta e com o algoritmo de *publish/subscribe*, ambos descritos na secção 7.2 deste relatório.

Numa fase inicial desta investigação foi determinado, através de simulações, que a distância máxima de comunicação, sem perda de pacotes, era de 120 metros. Considerando que, aumentando esta distância, se aumentava a probabilidade de perda de pacotes, foi considerado 120 metros como o raio efetivo de comunicação.

Na Tabela 6 são apresentadas as densidade de veículos por quilómetro quadrado, que são dadas pela fórmula: $\rho = \frac{N}{A}$, sendo "N" o número de nós e "A" a área em que os nós se movem. Na Tabela 7 são apresentados os cálculos da densidade média de veículos por quilómetro quadrado, bem como o número médio de veículos ao alcance de um rádio 802.11p. A formula para o número de veículos ao alcance de um rádio 802.11p é dada por: $X = \frac{N.\pi.r^2}{A}$, sendo a variável "N" o número de nós, "r" é o raio efetivo de comunicação e "A" a área onde os nós de movem.

8.2.1 Disseminação de dados usando algoritmo de pedido/resposta

As simulações efetuadas usando este algoritmo pretendiam comparar a execução do mesmo em diferentes trajetórias dos veículos, com diferentes números de *streams* subscritas, com percentagens variáveis de veículos com LTE e diferentes modos de transmissão dos *chunks*.

Assim, foram definidos 3 cenários de mobilidade distintos, que podem ser visualizados em mais detalhe na Tabela 6:

Trajeto	Número de *streams* disponíveis	Probabilidade de subscrever cada *stream*	Densidade de veículos por km^2	Percentagem de veículos com LTE
Edifício CISTER – Vila Nova de Gaia	1	100%	76	50%,30%,10%,5%
Edifício CISTER – Vila Nova de Gaia	2	70%		
Baixa do Porto - Norteshopping	1	100%	306	5%
Baixa do Porto – Norteshopping e Norteshopping – Baixa do Porto	1	100%	153	5%

Tabela 6 – Configurações dos cenários simulados

Todos os cenários foram simulados durante 1200 segundos (20 minutos), com um total de 240 nós veiculares. Todas as transmissões de *chunks* e de *advertisements* foram efetuadas recorrendo ao domínio de *broadcast*, com a exceção de uma simulação, que efetua a comunicação de *chunks*, apenas, em *unicast*. O último cenário descrito na Tabela 6 diz respeito a um cenário em que metade dos nós se encontra a percorrer o caminho Baixa do Porto – Norteshopping e a outra metade está a fazer o percurso inverso.

O modelo de mobilidade utilizado para os nós veiculares foi o `RoutesMobilityModel`, sendo que os pontos de partida e de chegada escolhidos, nos cenários CISTER – Vila Nova de Gaia, correspondem a locais reais contidos numa área circular de 1 quilómetro com centro no CISTER e no centro da cidade de Vila Nova de Gaia. Nos cenários Baixa do Porto – Norteshopping, os pontos de partida e de chegada selecionados correspondem a locais reais contidos numa área circular de 500 metros à volta da baixa do Porto e do Norteshopping.

Assim, foi possível avaliar os requisitos mínimos para o bom funcionamento do algoritmo, nomeadamente o número mínimo de veículos que necessitam de possuir os dados no início da simulação e a densidade de veículos necessária. Foi ainda possível comparar o desempenho do algoritmo com a transmissão de *chunks* em *broadcast* ou em *unicast*.

A Figura 63 e a Figura 64 mostram uma função distribuição acumulada, que relaciona as transmissões de *streams* completas com o tempo de simulação despendido, para os cenários CISTER – Vila Nova de Gaia.

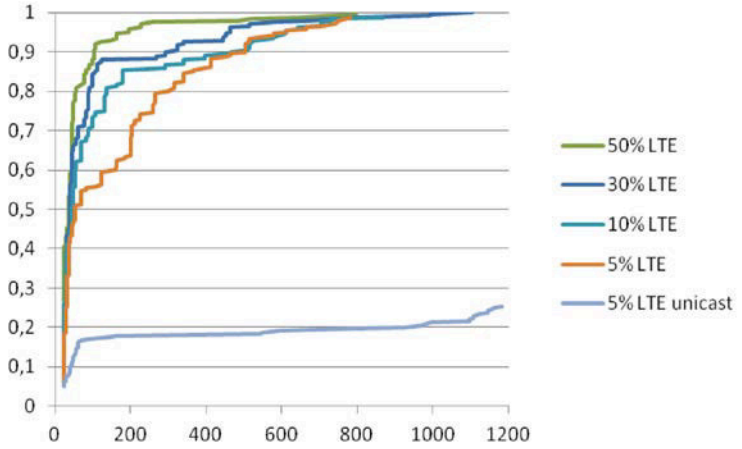


Figura 63- Função distribuição acumulada para o cenário CISTER - Vila Nova de Gaia com uma stream subscrita

Tal como é possível observar nas figuras, em ambos os casos, o efeito da transmissão dos *chunks* em *unicast* tem um impacto significativo no desempenho do algoritmo implementado. Por outro lado, nas simulações em que foi usado o domínio de *broadcast* para a disseminação dos *chunks*, a totalidade (ou, em alguns casos, perto da totalidade) dos nós foi capaz de receber toda a informação, mesmo em simulações com baixas concentrações de nós LTE.

No caso particular do cenário representado na Figura 64, é possível observar que na simulação

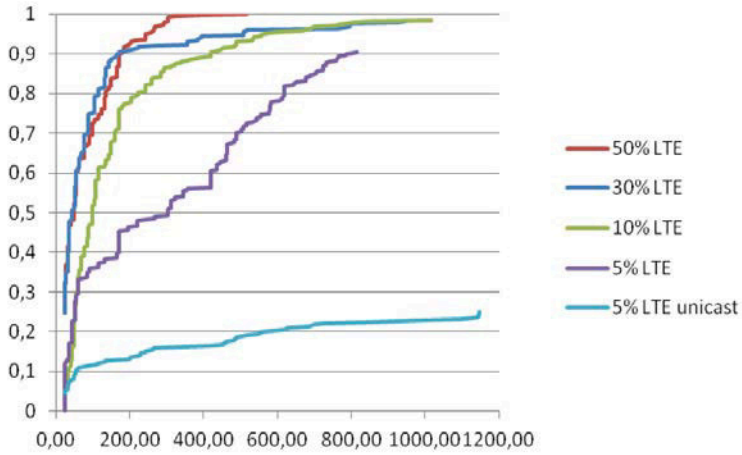


Figura 64 - Função de distribuição acumulada para o cenário CISTER-Vila Nova de Gaia com duas streams subscritas

de 5% de veículos LTE, a transmissão, ainda que em *broadcast*, dos *chunks* não é total, situando-se na zona dos 90%. Isto deve-se ao facto de os nós subscreverem as 2 *streams* com

a probabilidade de 70%, o que na prática significa que, enquanto no cenário anterior existiam 12 veículos com capacidades LTE, neste cenário apenas existem 8 veículos com capacidade LTE, em média, subscritos a cada *stream*.

É, também, possível observar que nas simulações com percentagens mais elevadas de nós LTE, os restantes nós recebem mais rapidamente toda a informação necessária. No entanto, podemos também afirmar que, desde que existam alguns nós na rede com a informação necessária, a mesma será redistribuída epidemicamente com sucesso.

A Figura 65 mostra os resultados das simulações executadas para os cenários Baixa do Porto – Norteshopping. Os dois cenários ilustrados representam o trajeto Baixa do Porto – Norteshopping (DN) e o trajeto Baixa do Porto – Norteshopping e Norteshopping – Baixa do Porto (DND). Os resultados demonstram que, no primeiro cenário (DN), existe um elevado desempenho do algoritmo, embora apenas existam 5% de veículos com capacidade LTE. Este desempenho deve-se ao facto de a densidade de veículos neste cenário ser consideravelmente mais elevada do que nos cenários anteriormente em estudo. Desta forma, os veículos passam mais tempo em contacto, o que leva a uma melhor disseminação da informação. O desempenho do algoritmo na disseminação de *chunks* transmitidos em *unicast*, no entanto, mantem um desempenho abaixo do desejado, sendo que apenas 35%, aproximadamente, dos nós recebe toda a informação da *stream*.

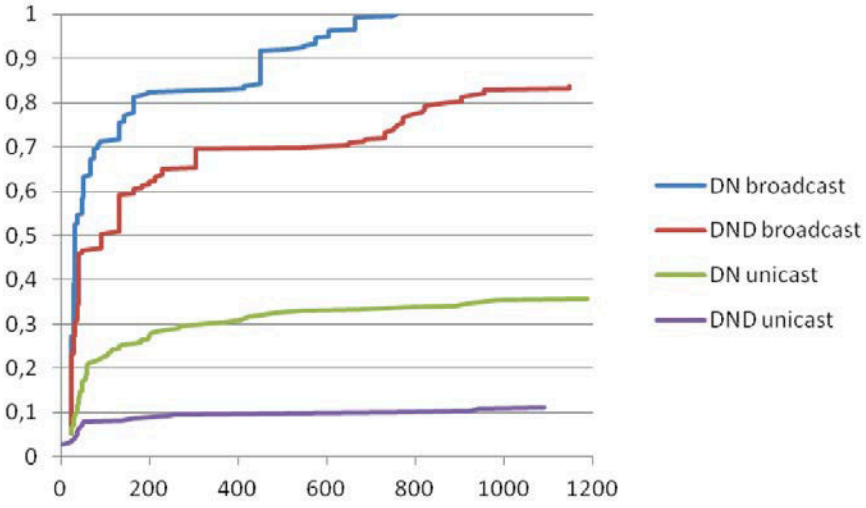


Figura 65 - Função de distribuição acumulada para os cenários Baixa do Porto - Norteshopping com uma stream subscrita

O segundo cenário (DND), no entanto, revela um desempenho mais fraco do algoritmo, tanto com a comunicação dos *chunks* a ser efetuada em *broadcast* como em *unicast*. Neste caso, como metade dos nós esta a fazer o percurso Baixa do Porto – Norteshopping e a outra metade

está a fazer o percurso inverso, a densidade média de veículos é de 153 veículos por quilómetro quadrado (no cenário DN, a densidade era de 306 veículos por quilómetro quadrado), o que provoca um impacto significativo no algoritmo de disseminação. Os resultados das simulações confirmam a hipótese de que o número de contactos entre os veículos nas duas áreas distintas é, aproximadamente, metade do existente no primeiro cenário (DN).

8.2.2 Disseminação de dados usando o algoritmo de *publish/subscribe*

Os cenários estudados para este algoritmo têm como objetivo a determinação da concentração, de veículos com capacidades LTE, e a densidade de total dos veículos com capacidades WAVE, necessárias para permitir uma disseminação total da informação. Os cenários em estudo consideram um número variável de nós, subscritos a apenas uma *stream*, cujas trajetórias foram escolhidas aleatoriamente, recorrendo ao `RoutesMobilityModel`, numa área de 2.25 quilómetros quadrados, no centro da cidade do Porto. Foram estudados cenários com 400, 200, 100, 80, 60, 40 e 20 nós, sendo que em cada um destes cenários foi estudado o efeito da variação do número de veículos com capacidades LTE. Para isso, foram efetuadas simulações, para cada cenário mencionado, com um veículo com LTE, com 5% e 10% de veículos com LTE. As simulações efetuadas tem uma duração de 300 segundos e fizeram uso do domínio de *broadcast* para a transmissão dos *chunks*, sendo os pedidos de *chunks* efetuados em *unicast*.

O objetivo do trabalho realizado pretende avaliar o desempenho do algoritmo implementado com diferentes densidades de veículos na rede, bem como variáveis percentagens de veículos equipados com tecnologia LTE, capaz de comunicação com a infraestrutura. Na Tabela 7 é possível ver a densidade média de veículos por quilómetro quadrado, e o número médio de veículos, ao alcance do rádio 802.11p, para cada cenário estudado.

Nós	400	200	100	80	60	40	20
Densidade de nós	177.78	88.89	44.44	35.56	26.67	17.78	8.89
Número médio de veículos ao alcance	8.042	4.021	2.011	1.608	1.206	0.804	0.402

Tabela 7 - Densidade de nós e número médio de veículos ao alcance, por quilómetro quadrado, para os cenários estudados

A Figura 66, Figura 67 e Figura 68 mostram as funções de distribuições acumuladas que relacionam as transmissões de *streams* completas com o tempo de simulação despendido. É

possível ver, em anexo, as funções de distribuições acumuladas para os restantes cenários estudados.

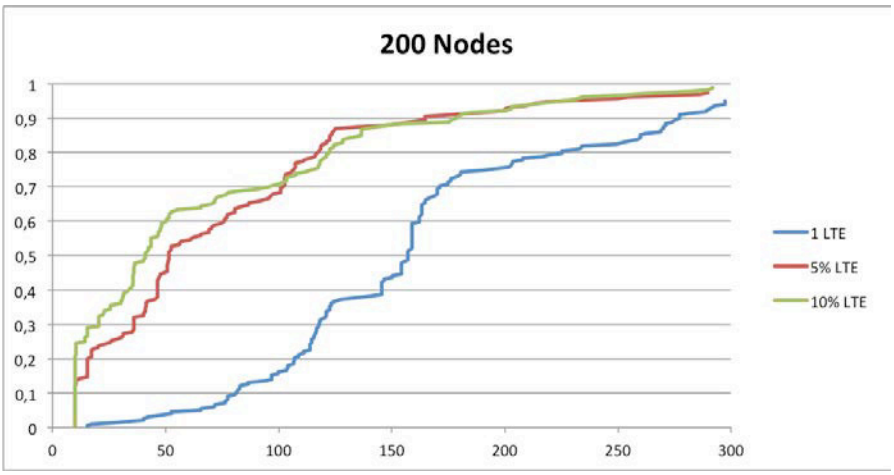


Figura 66 - Função de distribuição acumulada da receção total de uma stream, para o cenário de 200 nós

Tal como é possível observar nas figuras, a disseminação de informação é tão mais rápida quanto maior for a concentração de veículos que possuem a informação a disseminar, sendo possível observar que, ainda que exista apenas um veiculo a possuir a informação inicialmente, praticamente todos os veículos da rede acabam por receber a totalidade da informação, desde que existam nós suficientes na rede para disseminar a informação.

A Figura 66 ilustra, em particular, um cenário em que existem, aproximadamente, 4 nós ao alcance do rádio 802.11p, o que, na prática, se traduz num cenário de conexão praticamente total da rede, dadas as restrições espaciais da comunicação WAVE e a elevada mobilidade dos nós.

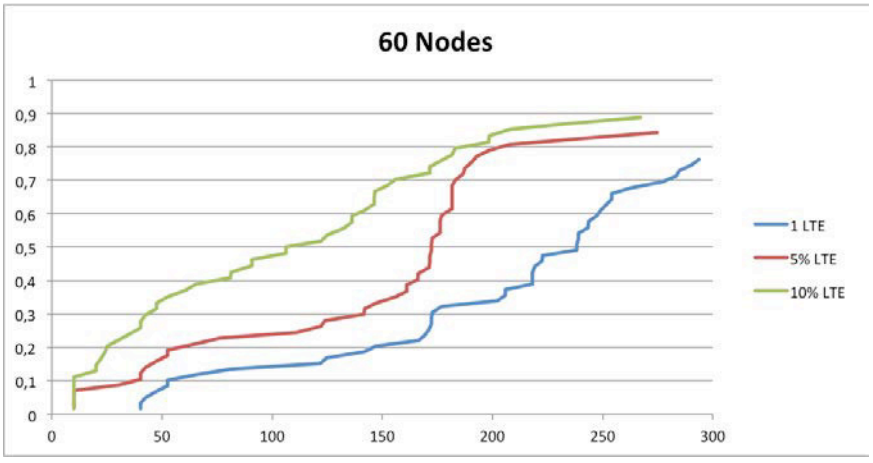


Figura 67 - Função de distribuição acumulada da receção total de uma stream, para o cenário de 60 nós

A Figura 67 mostra a disseminação dos dados de uma *stream* num cenário em que existe, em média, um nó ao alcance do rádio 802.11p. Neste cenário, é possível observar que o algoritmo apresentado tem um menor desempenho em situações de baixa concentração de nós na rede. Neste caso, os nós atingem um valor entre os 90% e os 78% de receção total da *stream*, contudo o tempo necessário para o atingir uma concentração acima de 50% varia entre, aproximadamente, 130 segundos e 250 segundos, o que não é, de todo, desejável para comunicações urgentes ou com restrições temporais elevadas.

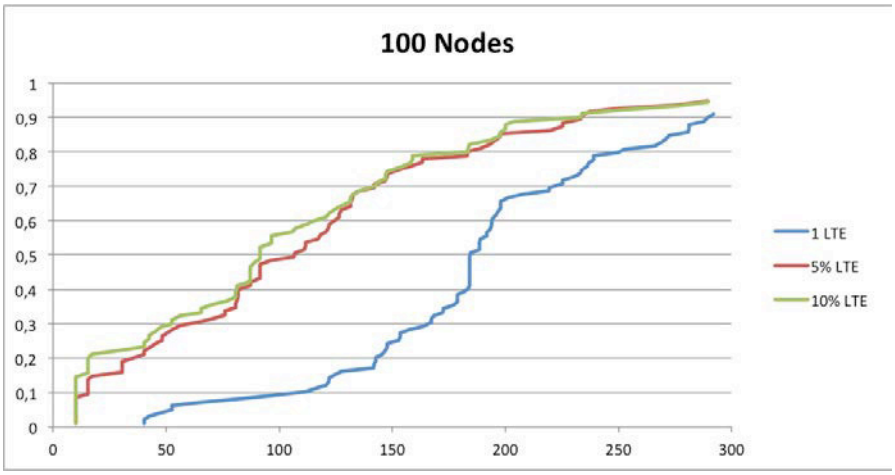


Figura 68 - Função de distribuição acumulada da receção total de uma stream, para o cenário de 100 nós

É, também, patente em todos os cenários que existe uma necessidade da rede possuir vários veículos com capacidades LTE, ou seja, veículos capazes de obter novas informações de *streams* e manter a informação existente na rede atualizada. Com efeito, em todas as simulações realizadas com apenas um nó LTE, mesmo as que contam com um elevado número de nós, existe um atraso claro, no início das simulações, na disseminação dos dados. Este comportamento deve-se, maioritariamente, ao facto de existirem, inicialmente, poucos nós a possuir a informação necessária e à medida que os nós vão recebendo informação, a velocidade com que os restantes nós recebem informação aumenta consideravelmente.

Adicionalmente, analisando as trocas de pacotes, é possível detetar uma taxa muito elevada de perda de pacotes nas simulações com elevada densidade de nós. Como é possível observar, na Figura 69, existe uma tendência para as simulações apresentarem uma elevada taxa de pacotes perdidos, sendo que as simulações que possuem apenas um nó LTE obtêm a menor taxa de pacotes perdidos. Este número elevado de perdas deve-se à colisão de pacotes, uma vez que existe uma grande densidade de nós nos cenários ilustrados. Da mesma forma, as simulações que contêm apenas um nó LTE possuem uma menor perda de pacotes por existirem menos nós capazes de responder a pedidos (que aumenta à medida que os nós

recebem os *chunks* da *stream* subscrita). Analisando estes dados em conjunto com os da Figura 66, é possível perceber a razão pela qual não existe uma disseminação mais célere em cenários de elevada densidade de nós, explicando também o aumento gradual da curva, representativa do cenário com apenas um nó LTE, bem como o seu subsequente abrandamento.

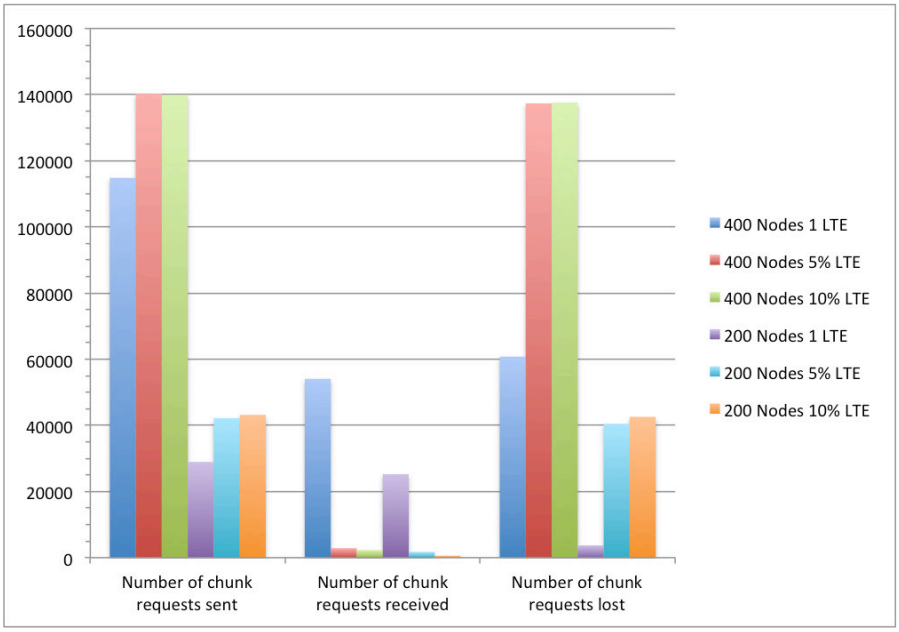


Figura 69 – Pedidos de chunks enviados, recebidos e perdidos para os cenários de 400 e 200 nós

A Figura 70 ilustra como, em cenários de menor densidade, a taxa de pedidos perdidos é significativamente menor, uma vez que a existência de uma menor densidade de nós traduz-se numa menor saturação do meio, desta forma resultando numa taxa de comunicação com baixa perda de pacotes. Da mesma forma, analisando estes resultados em conjunto com os resultados apresentados na Figura 67, é possível determinar a razão para um aumento mais gradual e constante da receção total dos dados contidos na *stream*, em comparação com cenários de maior densidade de nós.

Os problemas de saturação do meio, em redes veiculares, são conhecidos e tem sido propostas algumas abordagens ao problema [59], [60], que deverão ser estudadas num desenvolvimento futuro deste algoritmo.

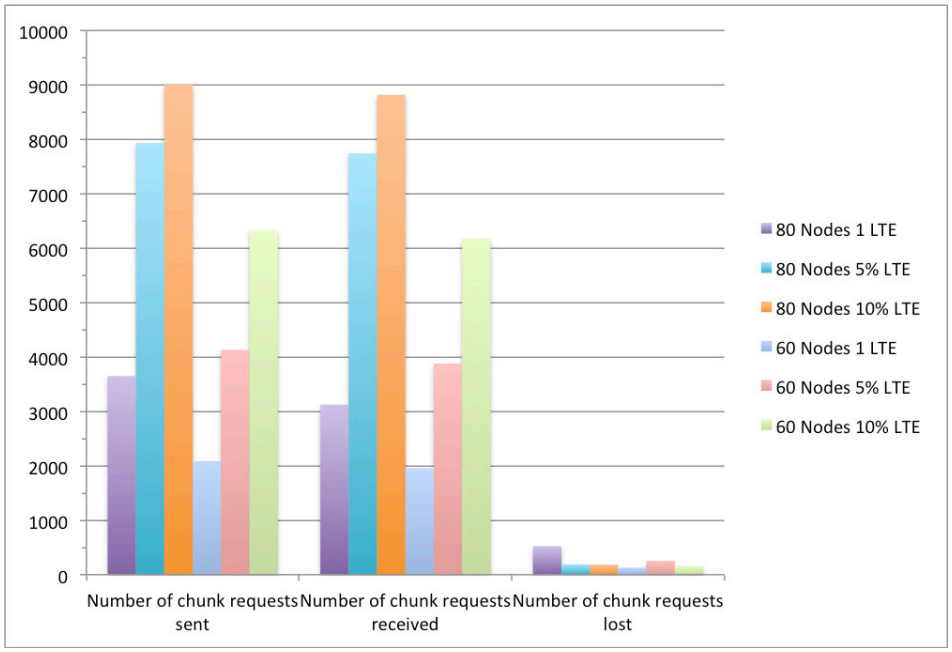


Figura 70 – Pedidos de chunks enviados, recebidos e perdidos para os cenários de 80 e 60 nós

9 Conclusões

Neste capítulo são apresentadas as conclusões finais relativamente a todo o trabalho desenvolvido neste projeto, assim como também será apresentado o sucesso ou insucesso no que diz respeito aos objetivos propostos. Serão, ainda, abordados outros trabalhos realizados no âmbito deste projeto, bem como identificadas as limitações do trabalho desenvolvido e possíveis trabalhos futuros.

9.1 Objetivos realizados

O objetivo principal deste projeto era a implementação de uma plataforma de disseminação de dados em redes veiculares, no simulador ns-3. De forma a alcançar esse objetivo e após uma análise ao problema proposto, foi decido implementar uma aplicação que fosse capaz de modelizar o comportamento de um servidor de *streams*, bem como uma aplicação capaz de simular os respetivos consumidores. Foram, então, propostos dois algoritmos de disseminação de dados, que foram implementados, com sucesso, nas aplicações descritas. Foi, ainda, elaborado um artigo científico para a conferência *World conference on Factory Communication Systems 2015* (WFCS), descrevendo o algoritmo de disseminação detalhado na secção 7.2.2.2 deste relatório.

No início do desenvolvimento deste projeto surgiu, também, a oportunidade de participação no *Google Summer of Code 2014*. Neste contexto, foi elaborada uma proposta de participação,

com um tema autoproposto, relacionado com o projeto de estágio a desenvolver. Esta proposta viria a ser o início do desenvolvimento do RoutesMobilityModel e, embora tenha sido rejeitada como projeto de *Summer of Code*, foi posteriormente aceite como um projeto de Verão tutorado por membros da equipa de desenvolvimento do ns-3. Este projeto de Verão tutorado previa, ainda, a oferta de uma bolsa para a participação na *Workshop on ns-3 2015* (WNS3), mediante a submissão, por parte do aluno, e posterior aceitação, por parte da equipa de revisores da conferência, de um artigo científico para a conferência WNS3, a realizar em Castelldefels, Barcelona.

Pode-se concluir, após revistos os objetivos inicialmente propostos, que todos os objetivos foram cumpridos com sucesso, uma vez que foram realizados estudos sobre o algoritmo de disseminação de dados implementado, que serviram para validar a implementação do mesmo. Da mesma forma, foi efetuado um trabalho semelhante no desenvolvimento do `RoutesMobilityModel`, tendo sido validada a implementação do mesmo. Foram ainda produzidos dois artigos científicos, um sobre o algoritmo de disseminação de dados [61] e outro sobre a implementação e validação do `RoutesMobilityModel` [32], que podem ser consultados em anexo, sendo este ultimo publicado e apresentado na conferência WNS3, pelo autor deste relatório.

9.2 Limitações e trabalho futuro

9.2.1 RoutesMobilityModel

Apesar de todos os objetivos propostos para o módulo de mobilidade terem sido cumpridos com sucesso, existe ainda espaço para melhoramentos neste módulo. A atual implementação, embora robusta e rica em funcionalidades pode beneficiar da adição de novos algoritmos de geração de mobilidade automática, bem como da adição de novos serviços de planeamento de viagens como, por exemplo, o *OpenStreetMaps*, e novas bases de dados de locais reais, desta forma ultrapassando algumas limitações das APIs implementadas atualmente como, por exemplo, a quota de utilização.

A atual implementação deste módulo possui um tempo computacional elevado, quando é aplicado a contentores com elevados números de nós. Este problema deve-se ao facto de ser necessário que o módulo efetue um *parse* da resposta XML enviada pelo Google Maps para, pelo menos, cada nó do contentor. No entanto, o tempo computacional despendido pelo módulo não rivaliza com o tempo necessário para a execução de uma simulação não trivial no ns-3. Este problema pode ser aliviado recorrendo ao uso de um *parser* JSON para a análise das respostas das APIs, contudo nunca será possível reduzir significativamente o tempo

computacional exigido para este módulo, uma vez que o tempo exigido tem uma relação direta com o número dos nós ao qual é aplicado.

O módulo poderia, ainda, beneficiar da implementação de algum tipo de serialização, permitindo ao utilizador repetir simulações com a mesma mobilidade. Esta funcionalidade necessita de um estudo prévio, de forma a integrar corretamente essa serialização com o restante código do ns-3. Para além disso, esta funcionalidade irá facilitar a repetição de simulações usando o módulo de mobilidade desenvolvido, na medida em que o tempo computacional exigido, em comparação com uma execução normal, será menor. Esta funcionalidade está já planeada para uma próxima versão do módulo.

Seria, ainda, benéfico a implementação de um método de interligação do módulo com a camada de rede do simulador, desta forma permitindo ao utilizador modelar aspetos de uma verdadeira rede veicular, por permitir aos nós que alterem a sua trajetória mediante uma dada informação recebida nas interfaces de rede. Por exemplo, no caso de ser recebido um alerta de engarrafamento para o trajeto planeado, existir um redireccionamento do percurso de forma a evitar a secção da estrada afetada. Esta funcionalidade tem o potencial para aumentar significativamente o realismo do módulo, e a sua implementação deve ser estudada.

Atualmente está planeado a escrita de um artigo, para submeter a uma revista científica da área de estudos, sobre o realismo do módulo desenvolvido, que irá visar comparações com mobilidade realística, previamente validada, bem como avaliar o desempenho do módulo com outras métricas.

9.2.2 Aplicação CarCoDe

Embora tenham sido cumpridos todos os objetivos propostos para o desenvolvimento da plataforma de comunicação veicular, esta pode ainda beneficiar de mais estudo, bem como da implementação de heurísticas que reduzam o problema de saturação do meio, tal como visto na secção 8.2.2. Assim, os algoritmos implementados devem ser comparados com outros já existentes, de forma a avaliar o desempenho do mesmo.

A implementação atual poderá, ainda, beneficiar do uso de protocolos de routing, tais como o AODV ou o OLSR, de forma a tirar proveito da comunicação *ad-hoc* para a transmissão de mensagens, de forma a reduzir o uso do domínio de *broadcast*. O impacto do uso destes protocolos deve, ainda, ser estudado no futuro.

Atualmente está a ser planeado a escrita de um artigo, para submissão a uma conferência da área de estudos, detalhando os resultados descritos na secção 8.2.2 deste relatório.

9.3 Contribuições

Sendo o ns-3 um projeto de *software* livre, as contribuições de código são um fator essencial ao desenvolvimento contínuo do mesmo e, como tal, as contribuições são fortemente encorajadas pela equipa. Como tal, no decorrer deste projeto, foram efetuados diversos contactos com a equipa do ns-3, no sentido de contribuir, essencialmente, o módulo de mobilidade desenvolvido. A versão estável do mesmo encontra-se, atualmente, em processo de revisão de código[62], de forma a ser incluída numa futura versão do ns-3.

Foi, ainda, efetuada uma participação num *sprint* de documentação, organizado pela equipa, em Outubro de 2014 [63], que visava, essencialmente, a melhoria da documentação de classes existente.

A aplicação CarCoDe, no entanto, não foi formalmente contribuída para o projeto. Contudo, o código implementado está licenciado ao abrigo da *Gnu General Public License version 2* (GPL) e encontra-se disponível no repositório oficial deste projeto de estágio [64].

Em resumo, este projeto permitiu:

- Desenvolver uma aplicação de disseminação de dados em ambientes veiculares, utilizando dois algoritmos distintos
- Implementar, testar e documentar um novo modelo de mobilidade realístico para o simulador de redes ns-3
- Participar, como autor, na conferência WNS3 2015, bem como interagir e contribuir para um projeto de *open-source* conhecido e ativamente mantido.

9.4 Outros trabalhos realizados

No âmbito deste projeto de estágio foram, ainda, realizadas algumas atividades complementares, que serão apresentadas de seguida.

9.4.1 Google Summer of Code 2014

Tal como referido anteriormente, foi elaborada uma proposta para um projeto de *Summer of Code*, que visava o desenvolvimento do módulo de mobilidade descrito neste relatório.

Esta proposta foi, no entanto, rejeitada como projeto de *Summer of Code*, uma vez que a equipa apenas podia atribuir quatro projetos, tendo optado por projetos propostos pela mesma. No entanto, e devido ao elevado número de candidatos apresentados, a equipa

ofereceu a oportunidade de realizar o projeto ao abrigo de um programa de Verão tutorado por membros da equipa do ns-3.

9.4.2 Participação na organização da conferência ARCS 2015

O CISTER organizou, em Março de 2014, a vigésima oitava conferência internacional em arquiteturas de sistemas computacionais e, devido à complexa logística da mesma, foi-me pedida ajuda para a organização. Desta forma, tive a oportunidade de participar em alguns seminários da conferência, bem como de participar na organização de uma conferência importante para a minha área de estudos.

9.4.3 Participação em seminários

O CISTER organiza, periodicamente, seminários para toda a equipa, que contam com a participação de oradores distintos. No espírito de aprendizagem e uma vez que me foi dada a oportunidade de assistir, fiz questão de estar presente naqueles cujos temas me suscitaram interesse, nomeadamente:

- Frank Mueller - Predictability for Uni- and Multi-Core Real-Time/Cyber-Physical Systems
- João Loureiro - XDense: A Dense Grid Sensor Network for Distributed Feature Extraction for Active Flow Control
- Hossein Fotouhi - Mobility Management in Low-Power Wireless Networks

9.4.4 Participação no *CISTER Periodic Seminar Series*

O CISTER organiza, também, seminários internos, que contam com a participação de membros da equipa de investigação, de forma a que estes possam mostrar o trabalho que tem vindo a desenvolver. Estes seminários servem para uma troca saudável de ideias, tendo ainda um forte componente social, na medida em que permitem conhecer as áreas de interesse e desenvolvimento de cada elemento.

Como membro da equipa, foi-me dada a oportunidade de assistir a estes eventos, bem como de participar, como orador. A apresentação realizada visou a implementação e modelação do modelo de mobilidade, bem como as suas funcionalidades e limitações.

9.4.5 Workshop on ns-3 (WNS3) 2015

No âmbito deste projeto foi elaborado um artigo científico sobre a modelação e implementação do `RoutesMobilityModel`. Este artigo viria a ser, posteriormente, aceite para publicação na conferência WNS3 2015.

Os projetos de Verão tutorados, no qual se enquadra o desenvolvimento do módulo de mobilidade descrito, contavam com uma bolsa de viagem, no valor de 350 euros, para todos os alunos que terminassem com sucesso o seu projeto e que possuíssem um artigo aceite na WNS3. Sendo este o meu caso, foi-me dada a oportunidade de representar o CISTER, acompanhado do meu orientador interno, na WNS3. Como tal, apresentei nessa conferencia o desenvolvimento do `RoutesMobilityModel`, bem como a sua modelação e validação do mesmo. Os diapositivos usados nesta apresentação podem ser visualizados em anexo.

Devido a questões logísticas foi-me, ainda, pedido que efetuasse a apresentação de três trabalhos em curso, da equipa de investigação do CISTER, aceites na conferência. Assim, apresentei, brevemente, os trabalhos:

- *A module for Data Centric storage in ns-3*[65]
- *A module for the XDense architecture in ns-3*[66]
- *A module for the FTT-SE protocol in ns-3*[67].

Adicionalmente, participei, em conjunto com o meu orientador interno, nas reuniões da equipa de desenvolvedores e na reunião do consórcio.

Em resumo, a minha participação na WNS3 2015 incluiu:

- Apresentação do artigo sobre o RoutesMobilityModel
- Breve apresentação dos trabalhos em curso, relacionados com o ns-3, desenvolvidos no CISTER
- Participação na reunião de desenvolvedores
- Participação na reunião do consórcio

9.4.6 Participação em *Sprint* de documentação

Tal como referido anteriormente, existiu uma participação num *sprint* de documentação, organizado pela equipa do ns-3, em Outubro de 2014. Este *sprint* teve como objetivo a eliminação de erros ou avisos gerados pelo Doxygen, que se relacionavam, principalmente, com funções ou variáveis não documentadas. Este *sprint* teve ainda um componente social, uma vez que permitiu, através de trabalho em conjunto, a interação com diferentes membros da equipa.

Esta intervenção teve um efeito positivo para o projeto, uma vez que resultou na eliminação de perto de 1800 avisos, no total[63].

10 Bibliografia

[1] Airbus Defence & Space, "Cassidian develops new applications for smart vehicle," *Cassidian develops new applications for smart vehicle*, Jul-2014. .

[2] G. Karagiannis, O. Altintas, E. Ekici, G. Heijenk, B. Jarupan, K. Lin, and T. Weil, "Vehicular Networking: A Survey and Tutorial on Requirements, Architectures, Challenges, Standards and Solutions," *IEEE Commun. Surv. Tutorials*, vol. 13, no. 4, pp. 584–616, 2011.

[3] J. Jakubiak and Y. Koucheryavy, "State of the art and research challenges for VANETs," in *Consumer Communications and Networking Conference, 2008. CCNC 2008. 5th IEEE*, 2008, pp. 912–916.

[4] A. K. Pandey, "Simulation of traffic movement in vanet using sumo," 2013.

[5] K. Mehta, L. G. Malik, and P. Bajaj, "VANET: Challenges, Issues and Solutions," 2013, pp. 78–79.

[6] A. Goulianos, N. F. Abdullah, D. Kong, M. Evangelos, D. Berkovskyy, A. Nix, and A. Doufexi, "Evaluation of 802.11 and LTE for Automotive Applications," 2014.

[7] M. Amadeo, C. Campolo, and A. Molinaro, "Enhancing IEEE 802.11p/WAVE to provide infotainment applications in VANETs," *Ad Hoc Networks*, vol. 10, no. 2, pp. 253–269, 2012.

[8] G. Araniti, C. Campolo, M. Condoluci, A. Iera, and A. Molinaro, "LTE for vehicular networking: A survey," *IEEE Commun. Mag.*, vol. 51, no. 5, pp. 148–157, 2013.

[9] D. Jiang and L. Delgrossi, "IEEE 802 . 11p : Towards an International Standard for Wireless Access in Vehicular Environments," *Group*, pp. 2036–2040, 2008.

[10] D. Denteneer, X. P. Costa, and N. E. C. L. Europe, "The IEEE 802 . 11 Universe," no. January, pp. 62–70, 2010.

[11] R. A. Uzcátegui and G. Acosta-Marum, "WAVE: A tutorial," no. December, 2009.

[12] S. a M. Ahmed, S. H. S. Ariffin, and N. Fisal, "Overview of wireless access in vehicular environment (wave) protocols and standards," *Indian J. Sci. Technol.*, vol. 6, no. 7, pp. 4994–5001, 2013.

[13] a Jafari and a Dogman, "Performance Evaluation of IEEE 802 . 11p for Vehicular Communication Networks," pp. 3–7, 2012.

[14] J. Matos, A. Oliveira, T. Meireles, and N. Ferreira, "Emergent Vehicular Communications : Applications , Standards and Implementation," no. 4.

[15] J. Bu, G. Tan, M. Liu, and C. Song, "Implementation and Evaluation of WAVE 1609.4/802.11p in ns-3."

[16] Q. Chen, D. Jiang, and L. Delgrossi, "IEEE 1609.4 DSRC multi-channel operations and its implications on vehicle safety communications," *2009 IEEE Veh. Netw. Conf. VNC 2009*, pp. 1–8, 2009.

[17] P. Sousa, "Non-IP multiprotocol for vehicular communications," Universidade de Aveiro, 2013.

[18] Y. L. Morgan, "Notes on DSRC & WAVE standards suite: Its architecture, design, and characteristics," *IEEE Commun. Surv. Tutorials*, vol. 12, no. 4, pp. 504–518, 2010.

[19] M. Hayashi, S. Fukuzawa, H. Ichikawa, and T. Kawato, "Development of Vehicular Communication (WAVE) System for Safety Applications," pp. 248–253, 2007.

[20] S. Sesia, I. Toufik, and M. Baker, *LTE – The UMTS Long Term Evolution*, vol. 3, no. 42. 2009.

[21] M. Sauter, *From GSM to LTE: An Introduction to Mobile Networks and Mobilie Broadband*. Wiley, 2011.

[22] C. Cox, *An Introduction to LTE: LTE, LTE-Advance, SAE and 4G Mobile Communications*, Second Edi. Wiley, 2014.

[23] 3GPP, *Evolved Universal Terrestrial Radio Access (E-UTRA) and Evolved Universal Terrestrial Radio Access Network (E-UTRAN); Overall Description*. 2013.

[24] 3GPP, *General Packet Radio Service (GPRS) Enhancements for Evolved Universal Terrestrial Radio Access Network (E-UTRAN) Access*. 2013.

[25] ns-3 consortium, "ns-3 Software Architecture," 2007.

[26] ns-3 consortium, "ns-3 Manual," 2014.

[27] ns-3 consortium, "ns-3 Documentation," 2015. [Online]. Available: https://www.nsnam.org/doxygen/index.html.

[28] ns-3 consortium, "ns-3 Tutorial," 2013.

[29] ns-3 consortium, "WAVE models — Model Library." [Online]. Available: https://www.nsnam.org/docs/models/html/wave.html. [Accessed: 11-Apr-2015].

[30] N. Baldo, "The ns-3 LTE Module Design Documentation."

[31] ns-3 consortium, “Mobility — Model Library.” [Online]. Available: https://www.nsnam.org/docs/models/html/mobility.html. [Accessed: 16-Apr-2015].

[32] T. Cerqueira and M. Albano, “RoutesMobilityModel : Easy Realistic Mobility Simulation using External Information Services,” in *Workshop on ns-3*, 2015, p. 7.

[33] F. Bai and A. Helmy, “A Survey of Mobility Models in Wireless Adhoc Networks,” *Wirel. Ad Hoc Sens. Networks*, pp. 1–30, 2004.

[34] F. J. Martinez, K. C. Toh, J. C. Cano, C. T. Calafate, and P. Manzoni, “A survey and comparative study of simulators for vehicularad hoc networks (VANETs),” no. October 2009, pp. 813–828, 2009.

[35] B. Boukenadil, “Importance of realistic mobility models for VANET networks,” *Int. J. Comput. Networks Commun.*, vol. 6, no. 5, pp. 175–182, 2014.

[36] F. K. Karnadi, Z. H. Mo, and K. C. Lan, “Rapid generation of realistic mobility models for VANET,” *IEEE Wirel. Commun. Netw. Conf. WCNC*, pp. 2508–2513, 2007.

[37] T. Camp, J. Boleng, and V. Davies, “A survey of mobility models for ad hoc network research,” *Wirel. Commun. Mob. Comput.*, vol. 2, no. 5, pp. 483–502, 2002.

[38] ns-3 consortium, “ns3::RandomWaypointMobilityModel Class Reference.” [Online]. Available: https://www.nsnam.org/docs/release/3.22/doxygen/classns3_1_1_random_waypoint_mobility_model.html#details. [Accessed: 17-Apr-2015].

[39] ns-3 consortium, “ns3::RandomWalk2dMobilityModel Class Reference.” [Online]. Available: https://www.nsnam.org/doxygen/classns3_1_1_random_walk2d_mobility_model.html#details. [Accessed: 17-Apr-2015].

[40] DLR - Institute of Transportation Systems, “SUMO – Simulation of Urban MObility.” [Online]. Available: http://www.dlr.de/ts/en/desktopdefault.aspx/tabid-9883/16931_read-41000/. [Accessed: 17-Apr-2015].

[41] DLR - Institute of Transportation Systems, “TraCI - Sumo.” [Online]. Available: http://sumo.dlr.de/wiki/TraCI. [Accessed: 17-Apr-2015].

[42] D. Krajzewicz, J. Erdmann, M. Behrisch, and L. Bieker, “Recent Development and Applications of SUMO – Simulation of Urban MObility,” vol. 5, no. 3, pp. 128–138, 2012.

[43] ns-3 consortium, “Contributing Code.” [Online]. Available: https://www.nsnam.org/developers/contributing-code/. [Accessed: 13-Jul-2015].

[44] Google, “Google Maps API Web Services.” [Online]. Available: https://developers.google.com/maps/documentation/webservices/. [Accessed: 17-Jun-2015].

[45] Google, “The Google Directions API.” [Online]. Available: https://developers.google.com/maps/documentation/directions/. [Accessed: 18-Jun-2015].

[46] Google, “Google Places API Web Service.” [Online]. Available: https://developers.google.com/places/webservice/intro. [Accessed: 18-Jun-2015].

[47] T. Cerqueira, “RoutesMobilityModel.” [Online]. Available: https://www.nsnam.org/wiki/RoutesMobilityModel. [Accessed: 21-Jun-2015].

[48] T. Cerqueira, “RoutesMobilityModel Archive 28-01-2015.” [Online]. Available: https://www.nsnam.org/mediawiki/index.php?title=RoutesMobilityModel&oldid=9272. [Accessed: 21-Jun-2015].

[49] P. Eeles, “Non-Functional Requirements,” 2005.

[50] C. Karney, “GeographicLib.” [Online]. Available: http://geographiclib.sourceforge.net/. [Accessed: 22-Jun-2015].

[51] Oracle, “When to use SAX.” [Online]. Available: https://docs.oracle.com/javase/tutorial/jaxp/sax/when.html. [Accessed: 25-Jun-2015].

[52] J. Henstridge, “Using the SAX interface of LibXML.” .

[53] Apache Software Foundation, “Xerces-C++ XML Parser.” [Online]. Available: https://xerces.apache.org/xerces-c/. [Accessed: 23-Jun-2015].

[54] T. Berners-Lee, R. T. Fielding, and L. Masinter, “RFC 3986,” 2005.

[55] Q. Ethan McCallum, “Processing XML with Xerces and SAX,” 2005. [Online]. Available: http://www.onlamp.com/pub/a/onlamp/2005/11/10/xerces_sax.html. [Accessed: 30-Jun-2015].

[56] Google, “Interactive Polyline Encoder Utility.” .

[57] C. Haerri, J. and Filali, F. and Bonnet, “Performance comparison of AODV and OLSR in VANETs urban environments under realistic mobility patterns,” *Proc. 5th IFIP Mediterr. Ad-Hoc Netw. Work.*, no. i, pp. 14–17, 2006.

[58] S. Robinson, “A statistical process control approach to selecting a warm-up period for a discrete-event simulation,” *Eur. J. Oper. Res.*, vol. 176, no. 1, pp. 332–346, 2007.

[59] M. Van Eenennaam, A. Remke, and G. Heijenk, “An analytical model for beaconing in VANETs,” *IEEE Veh. Netw. Conf. VNC*, pp. 9–16, 2012.

[60] O. K. Tonguz, N. Wisitpongphan, J. S. Parikh, F. Bai, P. Mudalige, and V. K. Sadekar, “On the broadcast storm problem in ad hoc wireless networks,” *2006 3rd Int. Conf. Broadband Commun. Networks Syst. BROADNETS 2006*, 2006.

[61] T. Cerqueira, M. Albano, and L. L. Ferreira, “Data Dissemination by Extending Publish / Subscribe to Vehicular Environments,” unpublished in *World Conference on Factory Communication Systems*, 2015, p. 7.

[62] T. Cerqueira, “Issue 265730044: Code review for the mobility-service-interface.” [Online]. Available: https://codereview.appspot.com/265730044/. [Accessed: 22-Sep-2015].

[63] ns-3 consortium, “Sprints.” [Online]. Available: https://www.nsnam.org/wiki/Sprints#Results_of_recent_sprints. [Accessed: 22-Sep-2015].

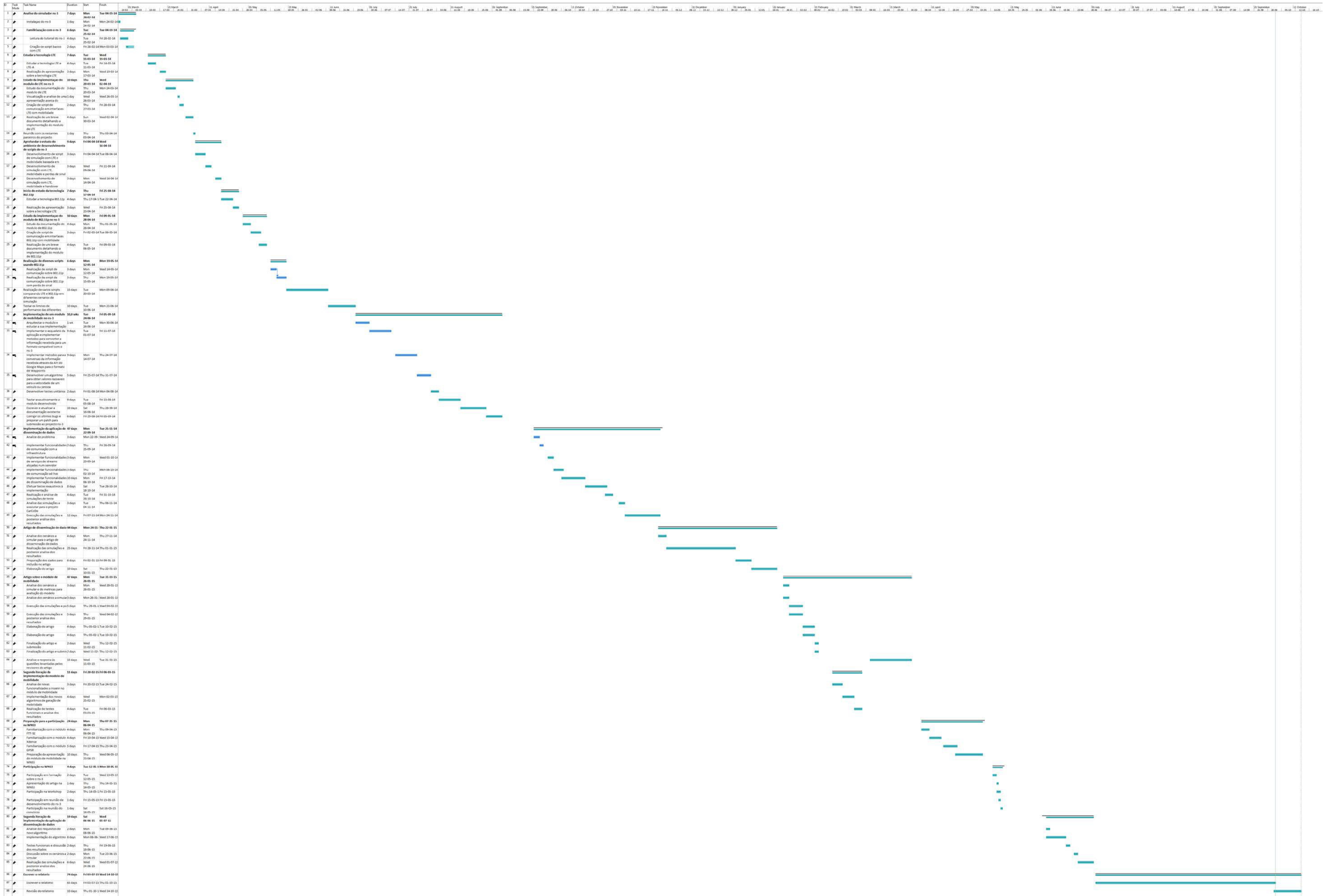
[64] T. Cerqueira, “PESTI.” [Online]. Available: https://github.com/Alcap/PESTI. [Accessed: 22-Sep-2015].

[65] M. Albano, T. Cerqueira, and S. Chessa, “A module for Data Centric storage in ns-3,” 2015.

[66] J. Loureiro, M. Albano, T. Cerqueira, R. Rangarajan, and E. Tovar, “A module for the XDense architecture in ns-3,” 2015.

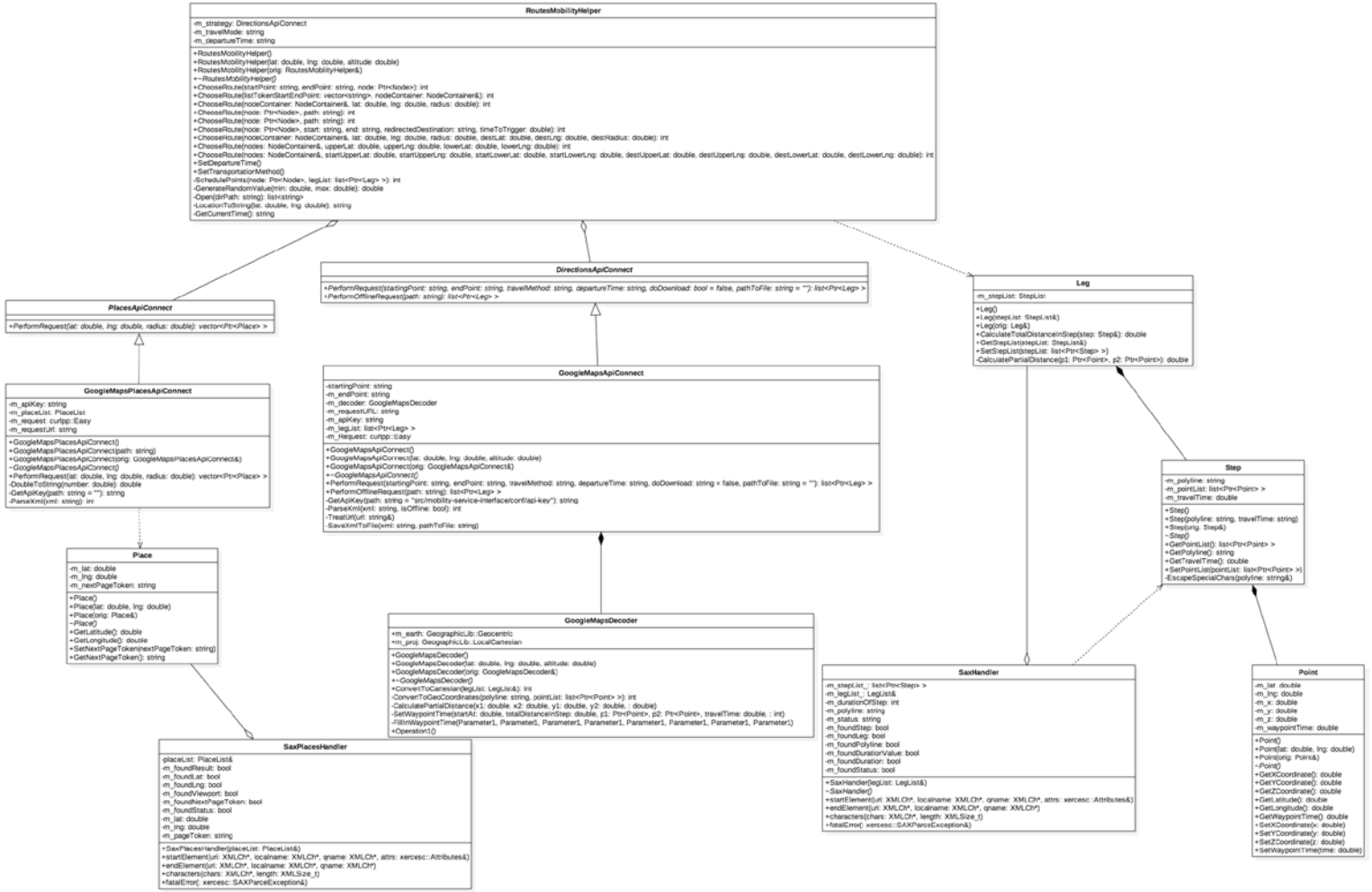
[67] F. Oliveira, R. Garibay-Martínez, T. Cerqueira, M. Albano, and L. L. Ferreira, “A module for the FTT-SE protocol in ns-3,” 2015.

Anexo 1 – Diagrama de Gantt do planeamento do projeto

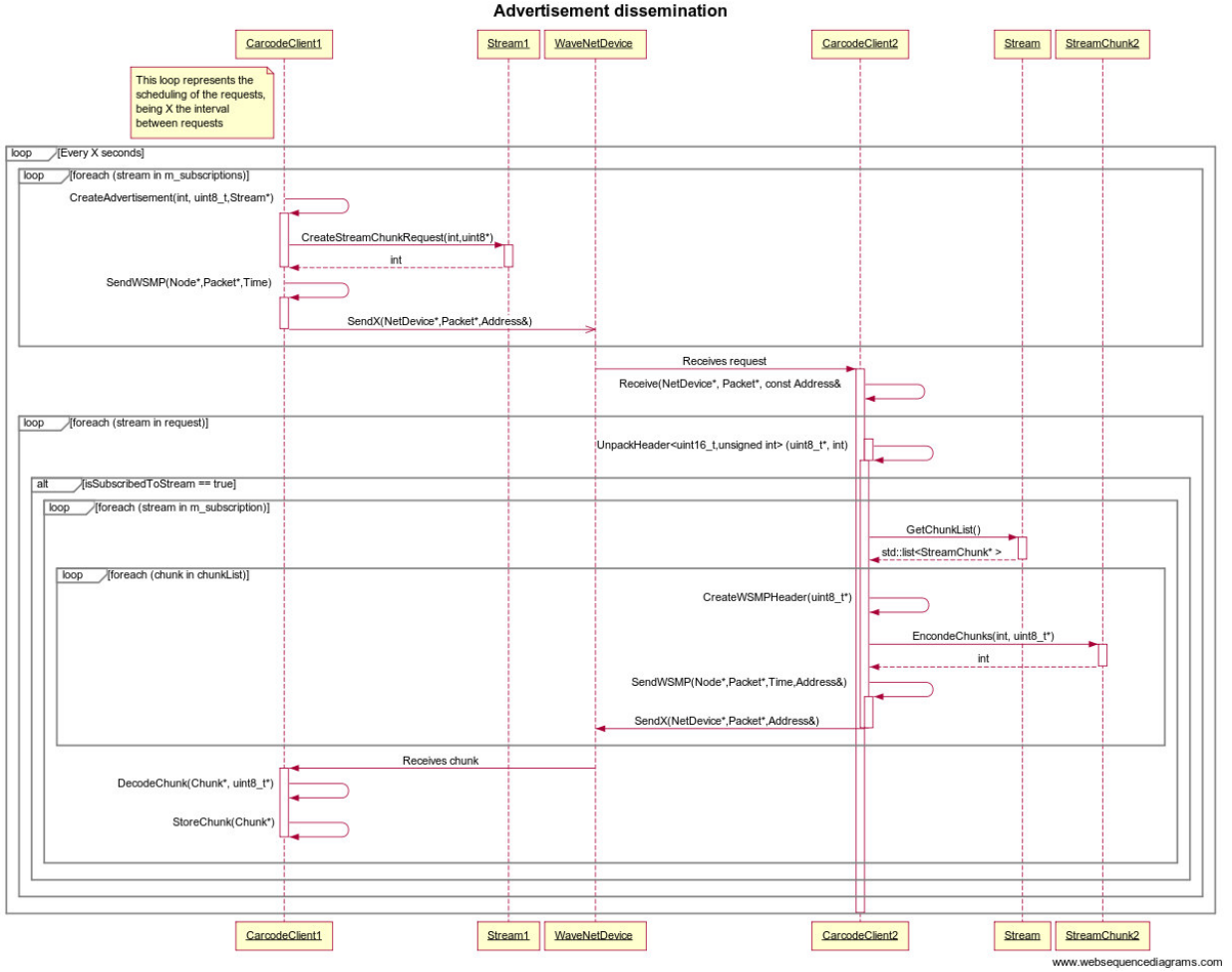


ID	Task Mode	Task Name	Duration	Start	Finish
1		**Analise do simulador ns-3**	**7 days**	**Mon 24-02-14**	**Tue 04-03-14**
2		Instalacao do ns-3	1 day	Mon 24-02-14	Mon 24-02-14
3		**Familiarizacao com o ns-3**	**6 days**	**Tue 25-02-14**	**Tue 04-03-14**
4		Leitura do tutorial do ns-3	4 days	Tue 25-02-14	Fri 28-02-14
5		Criação de script basico com LTE	2 days	Fri 28-02-14	Mon 03-03-14
6		**Estudar a tecnologia LTE**	**7 days**	**Tue 11-03-14**	**Wed 19-03-14**
7		Estudar a tecnologia LTE e LTE-A	4 days	Tue 11-03-14	Fri 14-03-14
8		Realização de apresentação sobre a tecnologia LTE	3 days	Mon 17-03-14	Wed 19-03-14
9		**Estudo da implementaçao do modulo de LTE no ns-3**	**10 days**	**Thu 20-03-14**	**Wed 02-04-14**
10		Estudo da documentação do modulo de LTE	3 days	Thu 20-03-14	Mon 24-03-14
11		Visualização e analise de uma apresentação acerca do	1 day	Wed 26-03-14	Wed 26-03-14
12		Criação de script de comunicação em interfaces LTE com mobilidade	2 days	Thu 27-03-14	Fri 28-03-14
13		Realização de um breve documento detalhando a implementação do modulo de LTE	4 days	Sun 30-03-14	Wed 02-04-14
14		Reunião com os restantes parceiros do projecto	1 day	Thu 03-04-14	Thu 03-04-14
15		**Aprofundar o estudo do ambiente de desenvolvimento de scripts do ns-3**	**9 days**	**Fri 04-04-14**	**Wed 16-04-14**
16		Desenvolvimento de script de simulação com LTE e mobilidade baseada em	3 days	Fri 04-04-14	Tue 08-04-14
17		Desenvolvimento de simulação com LTE, mobilidade e perdas de sinal	3 days	Wed 09-04-14	Fri 11-04-14
18		Desenvolvimento de simulação com LTE, mobilidade e handover	3 days	Mon 14-04-14	Wed 16-04-14
19		**Início do estudo da tecnologia 802.11p**	**7 days**	**Thu 17-04-14**	**Fri 25-04-14**
20		Estudar a tecnologia 802.11p	4 days	Thu 17-04-14	Tue 22-04-14
21		Realização de apresentação sobre a tecnologia LTE	3 days	Wed 23-04-14	Fri 25-04-14
22		**Estudo da implementaçao do modulo de 802.11p no ns-3**	**10 days**	**Mon 28-04-14**	**Fri 09-05-14**
23		Estudo da documentação do modulo de 802.11p	4 days	Mon 28-04-14	Thu 01-05-14
24		Criação de script de comunicação em interfaces 802.11p com mobilidade	3 days	Fri 02-05-14	Tue 06-05-14
25		Realização de um breve documento detalhando a implementação do modulo de 802.11p	4 days	Tue 06-05-14	Fri 09-05-14
26		**Realização de diversos scripts usando 802.11p**	**6 days**	**Mon 12-05-14**	**Mon 19-05-14**
27		Realização de script de comunicação sobre 802.11p	3 days	Mon 12-05-14	Wed 14-05-14
28		Realização de script de comunicação sobre 802.11p com perda de sinal	3 days	Thu 15-05-14	Mon 19-05-14
29		Realização de varios scripts comparando LTE e 802.11p em diferentes cenarios de simulação	15 days	Tue 20-05-14	Mon 09-06-14
30		Testar os limites de performance das diferentes	10 days	Tue 10-06-14	Mon 23-06-14
31		**Implementação de um modulo de mobilidade no ns-3**	**35,8 wks**	**Tue 24-06-14**	**Fri 05-09-14**
32		Arquitectar o modulo e estudar a sua implementação	1 wk	Tue 24-06-14	Mon 30-06-14
33		Implementar o esqueleto da aplicação e implementar metodos para converter a informação necessária para um formato compativel com o ns-3	9 days	Tue 01-07-14	Fri 11-07-14
34		Implementar metodos para a conversao da informação recebida através da API do Google Maps para o formato de Waypoints	9 days	Mon 14-07-14	Thu 24-07-14
35		Desenvolver um algoritmo para obter valores razoaveis para a velocidade de um veiculo ou pessoa	5 days	Fri 25-07-14	Thu 31-07-14
36		Desenvolver testes unitários	2 days	Fri 01-08-14	Mon 04-08-14
37		Testar exaustivamente o modulo desenvolvido	9 days	Tue 05-08-14	Fri 15-08-14
38		Escrever e atualizar a documentação existente	10 days	Sat 16-08-14	Thu 28-08-14
39		Corrigir os ultimos bugs e preparar um patch para submissão ao projecto ns-3	6 days	Fri 29-08-14	Fri 05-09-14
40		**Implementação da aplicação de disseminação de dados**	**47 days**	**Mon 22-09-14**	**Tue 25-11-14**
41		Analise do problema	3 days	Mon 22-09-14	Wed 24-09-14
42		Implementar funcionalidades de comunicação com a infraestrutura	7 days	Thu 25-09-14	Fri 26-09-14
43		Implementar funcionalidades de servigos de streamo alojadas num servidor	3 days	Mon 29-09-14	Wed 01-10-14
44		Implementar funcionalidades de comunicação ad-hoc	3 days	Thu 02-10-14	Mon 06-10-14
45		Implementar funcionalidades de disseminação de dados	10 days	Mon 06-10-14	Fri 17-10-14
46		Efetuar testes exaustivos à implementação	8 days	Sat 18-10-14	Tue 28-10-14
47		Realização e e análise de simulações de teste	4 days	Tue 28-10-14	Fri 31-10-14
48		Analise das simulações a executar para o projeto CarCoDe	3 days	Tue 04-11-14	Thu 06-11-14
49		Execução das simulações e posterior análise dos resultados	12 days	Fri 07-11-14	Mon 24-11-14
50		**Artigo de disseminação de dados**	**44 days**	**Mon 24-11-14**	**Thu 22-01-15**
51		Analise dos cenários a simular para o artigo de disseminação de dados	4 days	Mon 24-11-14	Thu 27-11-14
52		Realização das simulações e posterior analise dos resultados	25 days	Fri 28-11-14	Thu 01-01-15
53		Preparação dos dados para inclusão no artigo	6 days	Fri 02-01-15	Fri 09-01-15
54		Elaboração do artigo	10 days	Sat 10-01-15	Thu 22-01-15
55		**Artigo sobre o modulo de mobilidade**	**47 days**	**Mon 26-01-15**	**Tue 31-03-15**
56		Analise dos cenários a simular e do metricas para avaliação do modelo	3 days	Mon 26-01-15	Wed 28-01-15
57		Analise dos cenários a simular	3 days	Mon 26-01-15	Wed 28-01-15
58		Execução das simulações e pos	5 days	Thu 29-01-15	Wed 04-02-15
59		Execução das simulações e posterior analise dos resultados	5 days	Thu 29-01-15	Wed 04-02-15
60		Elaboração do artigo	4 days	Thu 05-02-15	Tue 10-02-15
61		Elaboração do artigo	4 days	Thu 05-02-15	Tue 10-02-15
62		Finalização do artigo e submissão	2 days	Wed 11-02-15	Thu 12-02-15
63		Finalização do artigo e submiss	7 days	Wed 11-02-15	Thu 12-02-15
64		Analise e resposta às questões levantadas pelos revisores do artigo	13 days	Wed 11-03-15	Tue 31-03-15
65		**Segunda iteração de implementação do modelo de mobilidade**	**11 days**	**Fri 20-02-15**	**Fri 06-03-15**
66		Analise de novas funcionalidades a inserir no módulo de mobilidade	3 days	Fri 20-02-15	Tue 24-02-15
67		Implementação dos novos algoritmos de geração de mobilidade	4 days	Wed 25-02-15	Mon 02-03-15
68		Realização de testes funcionais e analise dos resultados	4 days	Tue 03-03-15	Fri 06-03-15
69		**Preparação para a participação no WNS3**	**24 days**	**Mon 06-04-15**	**Thu 07-05-15**
70		Familiarização com o módulo FTT-SE	4 days	Mon 06-04-15	Thu 09-04-15
71		Familiarização com o modulo Ktenue	4 days	Fri 10-04-15	Wed 15-04-15
72		Familiarização com o módulo GPSR	5 days	Fri 17-04-15	Thu 23-04-15
73		Preparação da apresentação do módulo de mobilidade na WNS3	10 days	Thu 23-04-15	Wed 06-05-15
74		**Participação na WNS3**	**4 days**	**Tue 12-05-15**	**Mon 18-05-15**
75		Participação em formação sobre o ns-3	2 days	Tue 12-05-15	Wed 13-05-15
76		Apresentação do artigo na WNS3	1 day	Thu 14-05-15	Thu 14-05-15
77		Participação na Workshop	2 days	Thu 14-05-15	Fri 15-05-15
78		Participação em reunião de desenvolvimento do ns-3	1 day	Fri 15-05-15	Fri 15-05-15
79		Participação na reunião do consorcio	1 day	Sat 16-05-15	Sat 16-05-15
80		**Segunda iteração da implementação da aplicação de disseminação de dados**	**19 days**	**Sat 06-06-15**	**Wed 01-07-15**
81		Analise dos requisitos do novo algoritmo	2 days	Mon 08-06-15	Tue 09-06-15
82		Implementação do algoritmo	8 days	Mon 08-06-15	Wed 17-06-15
83		Testes funcionais e discussão dos resultados	2 days	Thu 18-06-15	Fri 19-06-15
84		Discussão sobre os cenários a simular	2 days	Mon 22-06-15	Tue 23-06-15
85		Realização das simulações e posterior analise dos resultados	6 days	Wed 24-06-15	Wed 01-07-15
86		**Escrever o relatorio**	**74 days**	**Fri 03-07-15**	**Wed 14-10-15**
87		Escrever o relatorio	65 days	Fri 03-07-15	Thu 01-10-15
88		Revisão do relatorio	10 days	Thu 01-10-15	Wed 14-10-15

Anexo 2 – Diagrama de classes para o RoutesMobilityModel


RoutesMobilityHelper
-m_strategy: DirectionsApiConnect
-m_travelMode: string
-m_departureTime: string
+RoutesMobilityHelper()
+RoutesMobilityHelper(lat: double, lng: double, altitude: double)
+RoutesMobilityHelper(orig: RoutesMobilityHelper&)
+~RoutesMobilityHelper()
+ChooseRoute(startPoint: string, endPoint: string, node: Ptr<Node>): int
+ChooseRoute(listTokenStartEndPoint: vector<string>, nodeContainer: NodeContainer&): int
+ChooseRoute(nodeContainer: NodeContainer&, lat: double, lng: double, radius: double): int
+ChooseRoute(noce: Ptr<Node>, path: string): int
+ChooseRoute(noce: Ptr<Node>, path: string): int
+ChooseRoute(noce: Ptr<Node>, start: string, end: string, redirectedDestination: string, timeToTrigger: double): int
+ChooseRoute(nodeContainer: NodeContainer&, lat: double, lng: double, radius: double, destLat: double, destLng: double, destRadius: double): int
+ChooseRoute(noces: NodeContainer&, upperLat: double, upperLng: double, lowerLat: double, lowerLng: double): int
+ChooseRoute(noces: NodeContainer&, startUpperLat: double, startUpperLng: double, startLowerLat: double, startLowerLng: double, destUpperLat: double, destUpperLng: double, destLowerLat: double, destLowerLng: double): int
+SetDepartureTime()
+SetTransportationMethod()
-SchedulePoints(node: Ptr<Node>, legList: list<Ptr<Leg> >): int
-GenerateRandomValue(min: double, max: double): double
-Open(dirPath: string): list<string>
-LocationToString(lat: double, lng: double): string
-GetCurrentTime(): string
DirectionsApiConnect
+PerformRequest(startingPoint: string, endPoint: string, travelMethod: string, departureTime: string, doDownload: bool = false, pathToFile: string = ""): list<Ptr<Leg> >
+PerformOfflineRequest(path: string): list<Ptr<Leg> >
Leg
-m_stepList: StepList
+Leg()
+Leg(stepList: StepList&)
+Leg(orig: Leg&)
+CalculateTotalDistanceInStep(step: Step&): double
+GetStepList(stepList: StepList&)
+SetStepList(stepList: list<Ptr<Step> >)
-CalculatePartialDistance(p1: Ptr<Point>, p2: Ptr<Point>): double
PlacesApiConnect
+PerformRequest(lat: double, lng: double, radius: double): vector<Ptr<Place> >
GoogleMapsApiConnect
-startingPoint: string
-m_endPoint: string
-m_decoder: GoogleMapsDecoder
-m_requestURL: string
-m_apiKey: string
-m_legList: list<Ptr<Leg> >
-m_request: curlpp::Easy
+GoogleMapsApiConnect()
+GoogleMapsApiConnect(lat: double, lng: double, altitude: double)
+GoogleMapsApiConnect(orig: GoogleMapsApiConnect&)
+~GoogleMapsApiConnect()
+PerformRequest(startingPoint: string, endPoint: string, travelMethod: string, departureTime: string, doDownload: string = false, pathToFile: string = ""): list<Ptr<Leg> >
+PerformOfflineRequest(path: string): list<Ptr<Leg> >
-GetApiKey(path: string = "src/mobility-service-interface/conf/api-key"): string
-ParseXml(xml: string, isOffline: bool): int
-TreatUrl(url: string&)
-SaveXmlToFile(xml: string, pathToFile: string)
GoogleMapsPlacesApiConnect
-m_apiKey: string
-m_placeList: PlaceList
-m_request: curlpp::Easy
-m_requestUrl: string
+GoogleMapsPlacesApiConnect()
+GoogleMapsPlacesApiConnect(path: string)
+GoogleMapsPlacesApiConnect(orig: GoogleMapsPlacesApiConnect&)
+~GoogleMapsPlacesApiConnect()
+PerformRequest(lat: double, lng: double, radius: double): vector<Ptr<Place> >
-DoubleToString(number: double): double
-GetApiKey(path: string = ""): string
-ParseXml(xml: string): int
Step
-m_polyline: string
-m_pointList: list<Ptr<Point> >
-m_travelTime: double
+Step()
+Step(polyline: string, travelTime: string)
+Step(orig: Step&)
+~Step()
+GetPointList(): list<Ptr<Point> >
+GetPolyline(): string
+GetTravelTime(): double
+SetPointList(pointList: list<Ptr<Point> >)
-EscapeSpecialChars(polyline: string&)
Place
-m_lat: double
-m_lng: double
-m_nextPageToken: string
+Place()
+Place(lat: double, lng: double)
+Place(orig: Place&)
+~Place()
+GetLatitude(): double
+GetLongitude(): double
+SetNextPageToken(nextPageToken: string)
+GetNextPageToken(): string
GoogleMapsDecoder
+m_earth: GeographicLib::Geocentric
+m_proj: GeographicLib::LocalCartesian
+GoogleMapsDecoder()
+GoogleMapsDecoder(lat: double, lng: double, altitude: double)
+GoogleMapsDecoder(orig: GoogleMapsDecoder&)
+~GoogleMapsDecoder()
+ConvertToCartesian(legList: LegList&): int
-ConvertToGeoCoordinates(polyline: string, pointList: list<Ptr<Point> >): int
-CalculatePartialDistance(x1: double, x2: double, y1: double, y2: double, : double)
-SetWaypointTime(startAt: double, totalDistanceInStep: double, p1: Ptr<Point>, p2: Ptr<Point>, travelTime: double, : int)
-FillInWaypointTime(Parameter1, Parameter1, Parameter1, Parameter1, Parameter1, Parameter1, Parameter1, Parameter1)
+Operation1()
SaxHandler
-m_stepList_: list<Ptr<Step> >
-m_legList_: LegList&
-m_durationOfStep: int
-m_polyline: string
-m_status: string
-m_foundStep: bool
-m_foundLeg: bool
-m_foundPolyline: bool
-m_foundDuratiorValue: bool
-m_foundDuration: bool
-m_foundStatus: bool
+SaxHandler(legList: LegList&)
~SaxHandler()
+startElement(uri: XMLCh*, localname: XMLCh*, qname: XMLCh*, attrs: xercesc::Attributes&)
+endElement(uri: XMLCh*, localname: XMLCh*, qname: XMLCh*)
+characters(chars: XMLCh*, length: XMLSize_t)
+fatalError(xercesc::SAXParseException&)
Point
-m_lat: double
-m_lng: double
-m_x: double
-m_y: double
-m_z: double
-m_waypointTime: double
+Point()
+Point(lat: double, lng: double)
+Point(orig: Point&)
+~Point()
+GetXCoordinate(): double
+GetYCoordinate(): double
+GetZCoordinate(): double
+GetLatitude(): double
+GetLongitude(): double
+GetWaypointTime(): double
+SetXCoordinate(x: double)
+SetYCoordinate(y: double)
+SetZCoordinate(z: double)
+SetWaypointTime(time: double)
SaxPlacesHandler
placeList: PlaceList&
-m_foundResult: bool
-m_foundLat: bool
-m_foundLng: bool
-m_foundViewport: bool
-m_foundNextPageToken: bool
-m_foundStatus: bool
-m_lat: double
-m_lng: double
-m_pageToken: string
+SaxPlacesHandler(placeList: PlaceList&)
+startElement(uri: XMLCh*, localname: XMLCh*, qname: XMLCh*, attrs: xercesc::Attributes&)
+endElement(uri: XMLCh*, localname: XMLCh*, qname: XMLCh*)
+characters(chars: XMLCh*, length: XMLSize_t)
+fatalError(xercesc::SAXParseException&)

Anexo 3 – Diagrama de sequencia completo para a disseminação de *advertisements*



Anexo 4 – Artigo publicado na WNS3 2015

Workshop on ns-3 - WNS3 2015 - ISBN: 978-1-4503-3375-7
Castelldefels (Barcelona), Spain - May 13-14, 2015

RoutesMobilityModel: Easy Realistic Mobility Simulation using External Information Services

Tiago Cerqueira
CISTER, ISEP/INESC-TEC
Rua Dr. António Bernardino de Almeida 431
4249-015, Porto, Portugal
1090678@isep.ipp.pt

Michele Albano
CISTER, ISEP/INESC-TEC
Rua Dr. António Bernardino de Almeida 431
4249-015, Porto, Portugal
mialb@isep.ipp.pt

ABSTRACT

The current implementation of ns-3 provides only synthetic mobility models that disregard the map where the nodes are moving, however, the study of vehicular ad-hoc networks requires the usage of more realistic mobility models. The usage of mobility traces created by traffic simulators such as SUMO is feasible, however, these simulators possess a steep learning curve, which prevents their fruition for most researchers whose research focus and expertise are on the data communication layer.

This paper presents a mobility model that generates realistic mobility traces that take into account the underlying maps, while maintaining the ease of usage that characterizes the synthetic mobility models. The module described herein is compared against SUMO and against the `ns3::RandomWaypointMobilityModel` of network simulator 3, to analyze the trade-off it implements in terms of realism and ease of usage.

Categories and Subject Descriptors

C.2.2 [**Network Protocols**]; I.6.5 [**Model development**]; I.6.7 [**Simulation support system**]

General Terms

Algorithms, Design, Experimentation

Keywords

Mobility model, trade-off analysis, network simulator 3

1. INTRODUCTION

Research and development activities in the fields of communication networks traditionally leverage simulations to explore the most probable results of the deployment of a given solution in a real scenario. Mobile communication is not an exception, and a plethora of simulators [1] were developed to cope with the problem of performing realistic simulations of mobile systems that communicate with each other and with a fixed infrastructure.

The research area of *Vehicular Ad-Hoc Networks* (VANETs) has seen accelerated development in recent years since the academia and the industry have produced specialized protocols [2], which have the potential to empower *vehicle-to-vehicle* (V2V) and *vehicle-to-infrastructure* (V2I) scenarios with communication capabilities, including safety related applications and traffic and fleet management. To satisfy research and development needs, a number of simulators have been either developed or adapted for vehicular communication simulation [3]. In order to correctly model these networks in a network simulator, a realistic mobility model must be used, as the mobility patterns have a significant impact on the vehicle's ability to communicate [4]. Current results have shown that synthetic mobility models, computed without taking into account the underlying road networks, can hardly be tailored to real maps and scenarios [5].

A few proposals have created vehicular traffic simulators [6], which are much more realistic and can be tailored to real-world maps. However, these simulators are quite complex to set up and need a solid experience in vehicular traffic simulation. Moreover, these traffic simulators are unable to simulate the effect of in-vehicle applications, which alter the vehicle's behaviour, unless Application Programming Interfaces (APIs) such as TraCI are used, adding yet another layer of complexity [7, 8].

This paper provides the description of a mobility model called RoutesMobilityModel that performs a trade-off between the synthetic mobility models, and complex vehicular traffic simulators. Our system accepts as inputs the details of the trajectory at large, connects to a travel planning service to download directions from the source to the destination of the planned trip, and converts it to a mobility trace for the simulator at hand. In particular, we implemented a prototype that makes use of Google Maps Service [9] in order to create mobility traces for the network simulator 3 (ns-3) [10].

The rest of the paper is structured as follows. Section 2 provides insights into existing mobility models, and the technologies we built upon while developing RoutesMobilityModel. Section 3 discusses the motivations that led us to develop RoutesMobilityModel. Section 4 describes the design and implementation of the module, Section 5 evaluates our module and the trade-off it implements between synthetic approaches and complex realistic approaches. Finally, Section 6 discusses limitations of the module and future work,

WNS3 2015, May 13 2015, Barcelona, Spain
© 2015 ACM ISBN 978-1-4503-3375-7/15/05 ...$15.00
DOI: http://dx.doi.org/10.1145/2756509.2756515.

Workshop on ns-3 - WNS3 2015 - ISBN: 978-1-4503-3375-7
Castelldefels (Barcelona), Spain - May 13-14, 2015

and Section 7 wraps up the paper.

2. RELATED WORK

This sections starts with a brief description of existing mobility models for vehicular networks, afterwards it describes the existing technologies that we built upon and integrated to design and implement the RoutesMobilityModel.

2.1 Existing Approaches

Currently, vehicular mobility is simulated either by using complex traffic simulators, such as Simulation of Urban Mobility (SUMO), or using synthetic mobility models, such as the random waypoint model, the random walk model or the Gauss-Markov model [3].

SUMO provides the most realistic mobility traces, however its higher complexity with respect to other mobility models (see for example [5]) makes it a burden to configure when the researcher's area of interest and expertise is on the data communication only. SUMO is capable of generating predefined abstract road networks or importing a digital road from services such as OpenStreetMaps or other traffic simulators such as VISUM, Vissim or MATSim. Creating a real-world road topology, however, is still a time consuming task, as configuration of complex scenarios, such as intersections, number of lanes, right of way, etc, still require a great deal of work [11].

Random mobility models, such as the random waypoint model and the Gauss-Markov mobility model are able to provide easy and fast mobility for network simulators, but their nature can often lead to unexpected results. In fact, these systems do not take the underlying maps into account, thus the realism of the mobility traces they generate is limited, and the importance of developing mobility models that take into account the underlying maps are testified by the multiple research activities in this area [3, 4, 5, 6].

The random waypoint model works by having every node in the simulation pick a random destination and a random velocity in certain parts of the trajectory. This, in turn, means that the nodes will travel at a constant velocity between the two parts of the trajectory (waypoints) [12]. Upon arrival at a waypoint, the node will pause for a determined amount of time and restart the process.

The Gauss-Markov model assigns speed and direction to a given node, and generates mobility by updating the node's speed and direction at a specified interval of time. This module is an improvement over other synthetic modules as it allows past velocities and directions to influence the future velocities and directions [13].

2.2 The Google Maps API

Our mobility model is based on the information made available by the Google Maps suite of services [9]. This subsection provides a brief description of the API to interact with the two services that are most interesting for our work, the Google Maps Direction API and the Google Maps Places API.

2.2.1 *The Google Maps Directions API*

The Google Maps Directions API offers a way for developers to interact with the Google Maps Directions Service, providing travel planning information based on transportation method. Our module makes use of this API in order to request from Google the path between two real world locations. Encoded polyline strings are returned by the service, representing movement between geographical coordinates (latitude and longitude) [14]. The API's response is articulated as follows:

- Leg – A travel is composed by Legs. Legs only occur if the user specifies a Waypoint (A to B, passing through C, for example), and each Leg is the trajectory to reach one Waypoint
- Step – A Leg is composed by Steps. A Step contains a polyline string, as well as the time estimation for the user to go from the first point of the polyline string to the last one, based on the transportation method chosen.

2.2.2 *The Google Maps Places API*

The Google Maps Places API returns information regarding particular establishments or point of interests, such as hospitals, gas stations, shops, etc. This information is used in tandem with the Google Maps Directions API, retrieving locations from a specified area, which are then used as start and endpoints in queries to the Directions API. This enables the user to quickly and effortlessly generate mobility for a node container, without having to manually specify start and end points.

The API response consists of a location (latitude and longitude) and information about the Place (such as its rating, its type, etc). The RoutesMobilityModel module uses the location attribute only.

2.3 The ns3::WaypointMobilityModel

Our mobility model produces traces that are imported into the ns-3 through its `ns3::WaypointMobilityModel` module. This subsection describes this module, to lay out the basis for the interaction of RoutesMobilityModel with ns-3.

The `ns3::WaypointMobilityModel` is capable of providing waypoint-based mobility to a node. Each object determines its velocity and position at a given time from a set of `ns3::Waypoint` objects. These objects are stored in a double ended queue, and are discarded when the current simulation time is greater than the time value of the object. When a node is in between waypoint times, it moves at a constant speed between the current and the previous waypoint [12].

The usage of this mobility module requires users to provide the Cartesian coordinates and the time of passage for each `ns3::Waypoint`, which is a cumbersome task to generate manually even for a simple mobility trace. On the other hand, `ns3::RoutesMobilityModel` generates these data programmatically, based on the XML traces returned by the Google Maps API.

2.4 External Libraries

A number of external libraries were integrated into the design of RoutesMobilityModel, to enable a faster and more robust fruition of the data returned by the Google Maps API.

2.4.1 *Xerces-C++*

Xerces-C++ [15] is a validating XML parser written in a portable subset of C++. This library is used in conjunction with the module described in order to parse the responses from the Google Maps APIs. The module uses the SAX2 parser, which is an event-driven mechanism for accessing XML documents.

Workshop on ns-3 - WNS3 2015 - ISBN: 978-1-4503-3375-7
Castelldefels (Barcelona), Spain - May 13-14, 2015

2.4.2 *GeographicLib*

GeographicLib is a small set of C++ classes for performing conversions between geographic, UTM, UPS, MGRS, geocentric and local Cartesian coordinates, for gravity, geoid height and geomagnetic field calculations. It is also used for solving geodesic problems [16]. The module makes use of the algorithms contained in GeographicLib library in order to, reliably, convert between World Geodetic System (WGS 84) coordinates and Cartesian coordinates.

2.4.3 *libcurl*

libcurl [17] is a free and easy-to-use client-side URL transfer library. The library supports, among others, the HTTPS protocol, which is required by the Google Maps APIs. This library is used to query the Google Maps APIs, via a regular HTTP request.

3. MOTIVATION

Simulations regarding Mobile Ad-hoc Networks (MANETs) and Vehicular Ad-hoc networks (VANETs) raise the need for proper mobility models. Many mobility models available in network simulators provide synthetic traces that disregard the maps the nodes are moving onto (see for example Random Walk 2D model, the Random Waypoint Model and the Gauss-Markov model). Research has shown that some aspects of vehicular traffic, such as acceleration and deceleration in the presence of obstacles, greatly affect the network performance [18]. Using traffic simulators such as SUMO within network simulators add realism, but a first time user with no experience with traffic simulators will encounter a steep learning curve. In order to generate a real-world road network, a SUMO user is required to run a handful of complex scripts, which require several configuration parameters each. The tutorial section in [11] provides a good reference to generate mobility traces ranging from the most basic to the most complex, however, it is quite lengthy and a user interested, mainly, in data communication is not likely to require such a high level of configuration.

The main goal of this paper is to describe a mobility model that is as easy to configure as the synthetic mobility models included within ns-3, while maintaining a high degree of realism, comparable to those offered by SUMO and similar software suites. Our module implements a trade-off between the two families of mobility models in terms of ease of usage and realism.

4. DESIGN AND IMPLEMENTATION

The module described in this paper provides an interface to convert information from a travel planning service to `ns3::Waypoints`, which are then used by ns-3 to manage the mobility of the simulated nodes. Currently, the only service accessible by `ns3::RoutesMobilityModel` is the Google Maps service [9]. The rationale behind this choice is that Google Maps service provides a robust and feature-rich API, which for example allows to select the mode of travel (walk, car, public transport). However, the design of our module allows easy addition of other services.

In order for the module to be useful for as many researchers as possible, the following use cases where taken into consideration while developing the module:

- Generation of mobility traces for a node, using either addresses or coordinates as start and end points

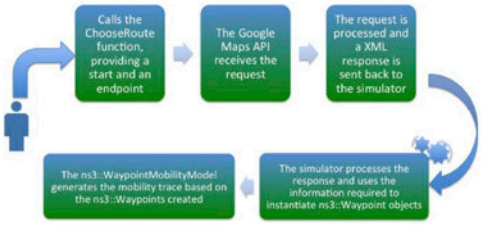


Figure 1: Module interaction when generating mobility traces

- Generation of mobility traces for all nodes contained into a `ns3::NodeContainer`
- Generation of mobility traces from previously downloaded responses
- Generation of mobility traces with dynamic node redirection

Our module uses the Google Maps Directions API that, given a start and an end point, computes the best path between them. The response of the API is then parsed by our module, in order to translate the response and create `ns3::Waypoints`. These waypoints are then imported in ns-3 through the `ns3::WaypointMobilityModel`, which processes the `ns3::Waypoints` and generates the corresponding mobility trace.

The usual interaction between the module, the simulator and the external information service is represented in Figure 1. The simulator invokes the `ns3::ChooseRoute` method, providing a start and an end point for the route. The RoutesMobilityModule contacts the Google Maps service, which answers with an XML file containing the route. The RoutesMobilityModule parses received data to create the list of waypoints corresponding to the route. Finally, the ns-3 simulator imports the waypoints to use them as the mobility pattern through its `ns3::WaypointMobilityModel` methods.

4.1 Module Architecture

The module described in this paper is articulated into several classes, which are responsible for the features described earlier in this section. The architecture is represented in Figure 2. Some of the most relevant classes include:

- RoutesMobilityHelper – helper class through which all of the features described in the paper are made available to the user. This class is responsible for creating `ns3::Waypoints` and adding them to the queue contained into the `ns3::WaypointMobilityModel` for further processing.
- GoogleMapsDecoder – responsible for decoding the polyline strings obtained from the Google Maps Directions API, into latitude and longitude pairs.
- GoogleMapsApiConnect and GoogleMapsPlacesApiConnect - responsible for querying the respective Google Maps APIs invoking the respective parsers for the retrieved XML files.

Workshop on ns-3 - WNS3 2015 - ISBN: 978-1-4503-3375-7
Castelldefels (Barcelona), Spain - May 13-14, 2015

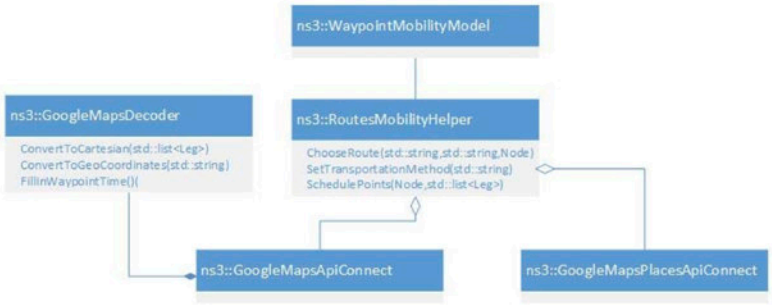


Figure 2: Architecture of the RoutesMobilityModel module

- SaxPlacesHandler and SaxHandler – responsible for parsing the Google Maps API's response.

The module was built in order to accommodate different travel planning services, such as OpenStreetMaps, and locations databases. To this end, the Strategy software design pattern was used in the module design. The Strategy pattern decouples the class' code from the algorithms it uses. This allows for the `ns3::RoutesMobilityHelper` to choose, at runtime, which travel planning service and location database to use.

The Strategy design pattern is also instrumental for easing up the implementation of new services that provide direction information or locations. In fact, the only required effort on part of the developer is the implementation of the provided interfaces and the usage of the provided model classes (`ns3::Leg`, `ns3::Step`, `ns3::Point` and `ns3::Place`) to represent the information retrieved by the implemented service.

4.2 Implementation

The RoutesMobilityModel module relies on the module `ns3::WaypointMobilityModel` in order to import the mobility routes retrieved from the external information services. Our module creates `ns3::Waypoints` that model the routes used to travel between two (or more) real world locations, and later on they are interpreted as the waypoints the vehicles move through during the simulation.

The information retrieved from the Google Maps Directions API contains, among other information, a polyline for each step of the route requested. It also contains the duration (the time it would take to go from the beginning to the end) for the step. The module decodes the polyline, thus creating a list of geographical coordinates, which are, in turn, converted to Cartesian coordinates. In order to model the speed of a node, implemented by setting the times for the `ns3::Waypoints`, we distributed the duration of the step in proportion to the distance traveled between two waypoints.

Three examples were also implemented, to provide users with additional information on the module and its usage:

- routes-mobility-example.cc – This example queries the Google Maps Directions API to generate a mobility trace based on a real-world route. The mobility traces generated in this example are from the city of Porto, Portugal
- routes-mobility-offline-example.cc – This example generates mobility traces based on Google Maps Directions API's XML files located on the hard disk. Before using the example, the user needs to download the file manually, as the Google Maps APIs Terms of Service forbid an application from caching the responses.
- routes-mobility-automatic-example.cc – This example uses the Google Maps Places API to generate mobility traces for node containers of up to 30 nodes. Mobility generation is done by querying the Google Maps Places API for places (restaurants, cinemas, etc), which will be used as start and end points. In this example, the area where the place are located is the city of Porto, Portugal.

5. VALIDATION

In order to validate the proposed mobility model, three scenarios were simulated, all of them featuring vehicles executing the same communication protocol between each other (the AODV routing protocol [18]), but with mobility generated using SUMO, the model `ns3::RoutesMobilityModel` and the model `ns3::RandomWaypointMobilityModel` respectively. This test leverages the test scripts that are part of the standard ns-3 distribution. Common parameters to the simulations were:

- 99 nodes
- 300 simulated seconds
- Vehicles randomly choose the route they take
- Nodes broadcast safety messages 10 times per second at 6 Mbps
- The TwoRayGround propagation model was used

A few parameters had to be different in the simulations, to take care of the structural diversity of the mobility models. To this end, the vanet-routing-compare.cc script was modified to simulate the following scenarios:

- SUMO scenario:

Workshop on ns-3 - WNS3 2015 - ISBN: 978-1-4503-3375-7
Castelldefels (Barcelona), Spain - May 13-14, 2015

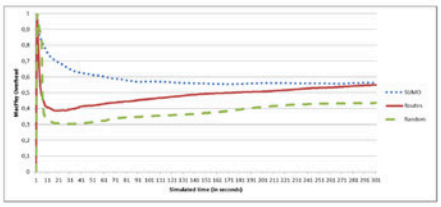


Figure 3: Overhead of the communication

- Mobility generated using SUMO for an area in downtown Barcelona, with width 4.6 km and height 3.0 km.

- RoutesMobility scenario:
  - Mobility in an area in downtown Barcelona, generated using the `ns3::RoutesMobilityModel`. Start and end points are obtained through a query to the Google Maps Places API. The radius requested was of 2.1 km, which leads to roughly the same area as the other scenarios.

- RandomWaypoint scenario:
  - Mobility generated in an area 4.6 km x 3.0 km using the `ns3::RandomWaypointMobilityModel`. We can consider the area to be located in downtown Barcelona.

The configuration of the SUMO scenario took a few hours, on the other hand both the `ns3::RoutesMobility` scenario and `ns3::RandomWaypoint` scenario were quite straightforward. In particular, RoutesMobilityModel required only the specifying of the kind of places to be retrieved, the center of the simulation area, and its radius.

The simulation results evaluated the communication overhead in terms of the ratio between the total bits sent on the wireless medium, and the payload net of MAC and PHY overhead. We conjectured that our mobility model led to AODV communication to presenting a performance similar to the case of SUMO mobility, while both being quite different from `ns3::RandomWaypointMobilityModel`. Simulation results, reported in Figure 3, correspond to the mean behavior of the AODV protocol over 10 simulations. The results confirm the convergence over time in the behavior of SUMO and `ns3::RoutesMobilityModel` and their difference from `ns3::RandomWaypointMobilityModel`, thus providing a hint regarding the relative realism of our module with respect to `ns3::RandomWaypointMobilityModel`.

The mobility traces generated are also quite similar in the SUMO scenario and in the RoutesMobility scenario. Figure 4 and Figure 5 show the mobility routes generated in the SUMO scenario and in the RoutesMobility scenario respectively, with the bullets represent initial location of vehicles, and the lines represent the routes taken by the vehicles according to the mobility model.

The mobility generated in the RandomWaypoint scenario, shown in Figure 6, is drastically different from the two mobility traces presented above.

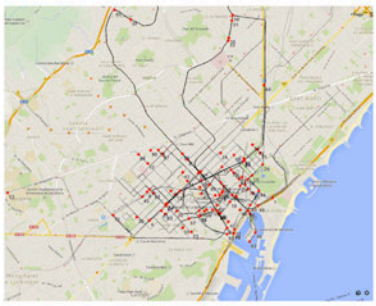


Figure 4: Mobility of SUMO scenario

Figure 5: Mobility of RoutesMobility scenario

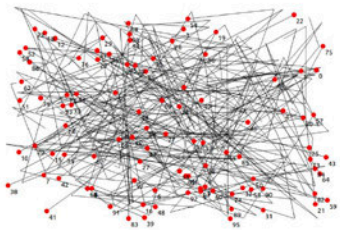


Figure 6: Mobility of RandomWaypoint scenario

The results shown in this section hint at the usefulness of the RoutesMobilityModel module to researchers studying VANETs or MANETs, as the module implements a nice trade-off between ease of usage and realism.

The mobility generated using the module described in this paper is accurate both in space and time, as it uses

Workshop on ns-3 - WNS3 2015 - ISBN: 978-1-4503-3375-7
Castelldefels (Barcelona), Spain - May 13-14, 2015

the partial trip time, as calculated by the Google Maps Directions API, to decide at which time it places a specific `ns3::Waypoint`. This, in practice, means that the nodes will always travel the speed Google expects a real car to travel through that particular road, taking into account kind of road, presence of roundabouts, sharp turns, onramps, etc. Moreover, if traffic information is available for the route chosen, the car will travel the sections under traffic at a realistic speed. This feature is only available using the "Google Maps APIs for Work" service, which is a pay service, otherwise traffic information is not included.

The RouteMobilityModel module is currently being used in simulative analysis of data dissemination algorithms in vehicular environments.

6. LIMITATIONS AND FUTURE WORK

Our current research work aims at developing further the module, since it still possesses room for improvement.

Firstly, the module will only be useful if it is able to provide mobility for a node container of a sufficient size. Currently, the Google Maps Places API is only able to return up to 60 locations, which greatly impairs this module. Because of this, the current implementation only supports automatic mobility generation for node containers of 30 nodes, however, several solutions for this problem have been proposed:

- Randomly choose a start and an end point from the places returned by the query to the Google Maps Places API.
- Use a different service which is capable of returning more than 60 locations.
- Generate routes by combining pairs of points taken from the same set, producing a number of routes that scale as the square of the number of points. A drawback of this approach is that the nodes would essentially keep traveling using many times the same streets / freeways, etc.
- Generate routes by randomly select a start and endpoint from a user-specified pool of locations
- Generate routes by combining pairs of points, the first one being chosen at random in a set of starts points, the second one chosen at random in a set of destination points. Similar to the previous one, this solutions scales quadratically as the number of points in the two sets.

The first proposed solution was tested successfully. The resulting mobility appears to be realistic with respect to typical behaviors of car drivers.

In an experimental version of the RoutesMobilityModel module, the fourth and fifth solutions were also tested. By defining manually the sets of points, it is possible to provide mobility for containers of any size. By using this technique in conjunction with the Google Places API, using the latter to create a pool of address to use as start and end points, it is possible to automatize the selection of points and still generate a large number of routes. The approaches mentioned in this paragraph were tested with positive results with 240 nodes (Figure 7 and Figure 8).

The computational time of the module is another drawback that must be addressed in the future. Currently, the module takes a long time to generate mobility for large node containers, since it is required to parse a large XML response per node in the container. The switch from an XML parser to a JSON parser can provide valuable performance enhancement, and it is a planned future work.

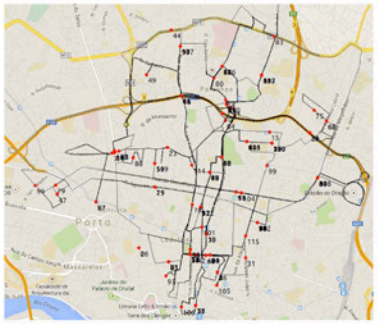


Figure 7: Mobility generated using user specified locations

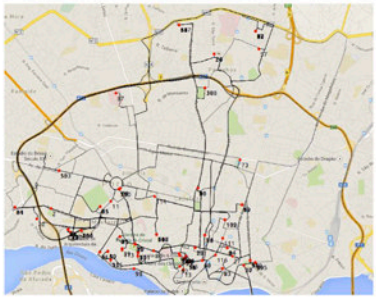


Figure 8: Mobility generated using the Google Maps Places API

Currently, the module is also lacking proper serialization capabilities, which is the capability of the module of storing on the filesystem the data returned by the external information service, with the aim of parsing the data to a ns-3 mobility model at a later time and potentially multiple times to guarantee repeteability of the simulation. A few efforts were made in the early stages of the development, however, no serialization is implemented at this time. This matter should be studied and implemented in the future, as proper serialization is key, both to improve the usability of the module by decreasing the computational time for creating the `ns3::Waypoints`, and to allow users to generate a simulation scenario once and load it as needed.

The module also lacks proper bidirectional coupling between the network simulator's network modules and the mobility model created. That is, the module does not react to

Workshop on ns-3 - WNS3 2015 - ISBN: 978-1-4503-3375-7
Castelldefels (Barcelona), Spain - May 13-14, 2015

the data a node receives. It should be possible for the users to specify actions such as redirecting a node based on data received, for example. We plan to develop an API to allow for this bidirectional coupling, empowering users to enable the nodes to make traffic routing decisions based on data received.

A last improvement we propose for our module aims at exploiting the data from travel planning services and location databases different from the Google Maps services.

Finally, regarding the investigation methodology, we have planned to perform a more proper comparison with other mobility models, both synthetic such as the Gauss-Markov model and realistic such as the multi-agent microscopic traffic simulator (MMTS) [19], and consider more metrics to evaluate the realism and ease of usage of our mobility model.

7. CONCLUSION

The work presented in this paper details a mobility model capable of generating mobility with an acceptable degree of realism, while maintaining the same (or even greater) ease of use that characterize the synthetic mobility models that do not take into account real-world maps. The work described was validated through simulation scenarios, with encouraging results. Results showed that, while RoutesMobilityModel is not as realistic as traffic simulators, it still maintains an acceptable level of realism for VANETs. MANET researchers will also be able to take advantage of this module, since Google Maps API allows to model movement for nodes traveling on foot or using public transportation.

8. ACKNOWLEDGMENTS

This work was supported by the Portuguese Agency for Innovation (ADI) under the ERDF (European Regional Development Fund) through COMPETE (Operational Programme 'Thematic Factors of Competitiveness'), within project CARCODE, ITEA2 Nr. 11037, QREN - SI I&DT Nr. 30345.

9. REFERENCES

[1] E. Weingartner, H. Vom Lehn, and K. Wehrle. "A performance comparison of recent network simulators." IEEE International Conference on Communications (ICC'09), 2009.

[2] Y.-A. Daraghmi, C.-W. Yi, and I. Stojmenovic. "Forwarding methods in data dissemination and routing protocols for vehicular ad hoc networks." Network, IEEE 27.6, pp. 74-79, 2013.

[3] F. J. Ros, J. A. Martinez, and P. M. Ruiz. "A survey on modeling and simulation of vehicular networks: Communications, mobility, and tools." Computer Communications 43, pp. 1-15, 2014.

[4] F. K. Karnadi, Z. H. Mo, and K. Lan. "Rapid generation of realistic mobility models for VANET." IEEE Wireless Communications and Networking Conference (WCNC 2007), 2007.

[5] F. J. Martinez, C. K. Toh, J. C. Cano, C. T. Calafate, and P. Manzoni. "A survey and comparative study of simulators for vehicular ad hoc networks (VANETs)." Wireless Communications and Mobile Computing 11.7, pp. 813-828, 2011.

[6] V. D. Khairnar and S. N. Pradhan. "Comparative study of simulation for vehicular ad-hoc network." arXiv preprint arXiv:1304.5181, 2013.

[7] D. Krajzewicz, J. Erdmann, M. Behrisch, and L. Bieker. "Recent development and applications of SUMO – simulation of urban mobility." International Journal on Advances in Systems and Measurements 5.3 and 4, pp. 128-138, 2012.

[8] C. Sommer, Z. Yao, R. German, and F. Dressler. "On the need for bidirectional coupling of road traffic microsimulation and network simulation." Proceedings of the 1st ACM SIGMOBILE workshop on Mobility models, 2008.

[9] C. C. Miller. "A beast in the field: The Google Maps mashup as GIS/2." Cartographica: The International Journal for Geographic Information and Geovisualization 41.3, pp. 187-199, 2006.

[10] T. R. Henderson, M. Lacage, G. F. Riley, C. Dowell, and J. B. Kopena. "Network simulations with the ns-3 simulator." SIGCOMM demonstration, 2008.

[11] "SUMO Wiki page", available online at: http://sumo.dlr.de/wiki/SUMO

[12] ns-3 Consortium. "ns3::RandomWaypointMobilityModel Class Reference" (2015), online at: http://www.nsnam.org/doxygen/classns3_1_1_random_waypoint_mobility_model.html

[13] T. Camp, J. Boleng, and V. Davies. "A survey of mobility models for ad hoc network research." Wireless communications and mobile computing 2.5, pp. 483-502, 2002.

[14] Developers' forum for Google Maps API, "Decoding polylines in Google Maps Directions API", available online at https://developers.google.com/maps/documentation/utilities/polylinealgorithm

[15] T. W. Leung. "Professional XML Development with Apache Tools: Xerces, Xalan, FOP, Cocoon, Axis, Xindice". John Wiley & Sons, 2004.

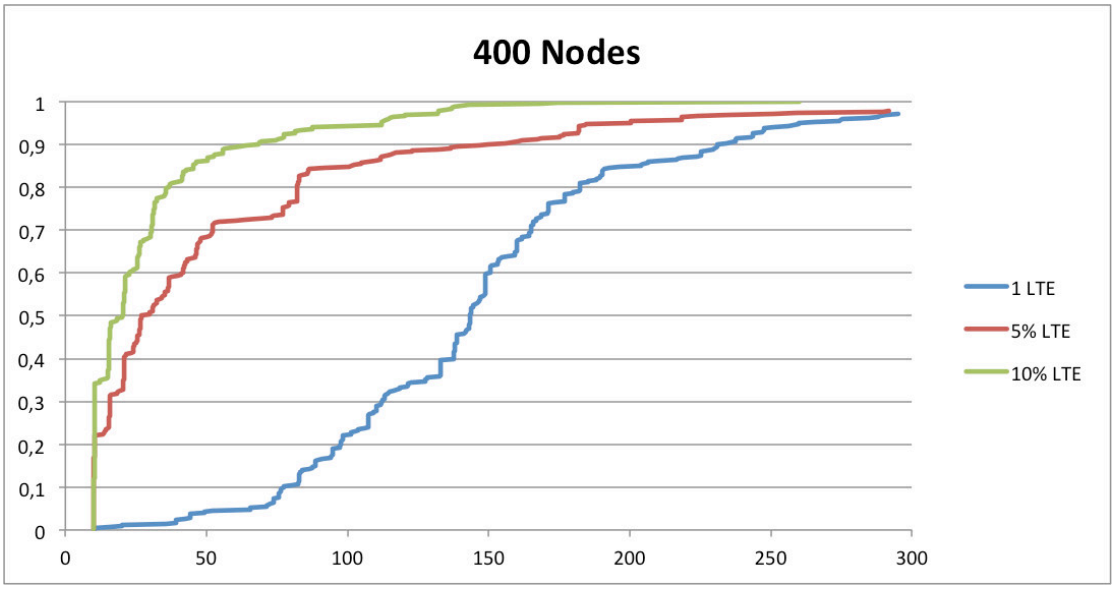
[16] C. Karney, "GeographicLib", online at http://geographiclib.sourceforge.net/

[17] D. Stenberg, "libcurl: The multiprotocol file transfer library", online at: http://curl.haxx.se/libcurl/

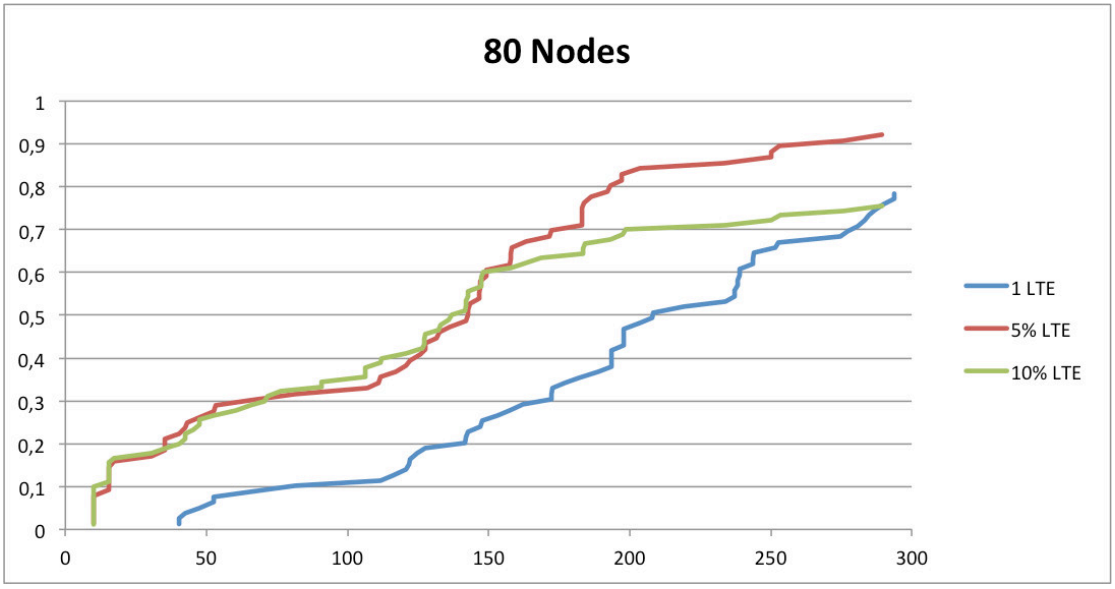
[18] J. Haerri, F. Filali, and C. Bonnet. "Performance comparison of AODV and OLSR in VANETs urban environments under realistic mobility patterns." Proceedings of the 5th IFIP mediterranean ad-hoc networking workshop, 2006.

[19] B. Raney, A. Voellmy, N. Cetin, M. Vrtic, and K. Nagel. "Towards a microscopic traffic simulation of all of Switzerland." The International Conference on Computational Science (ICCS 2002), Springer Berlin Heidelberg, pp. 371-380, 2002.

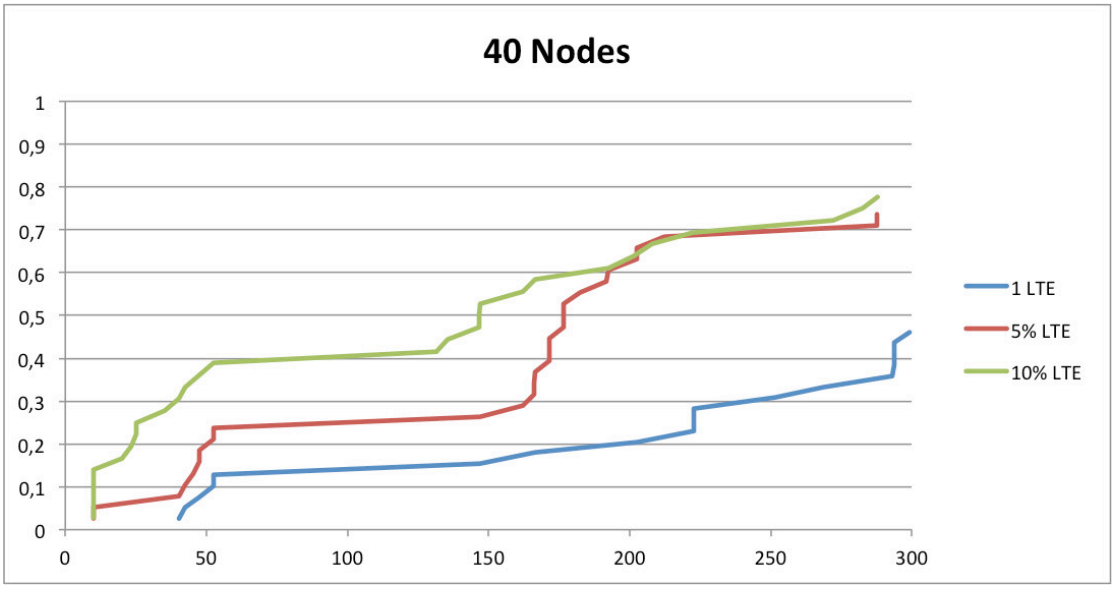
Anexo 5 - Função de distribuição acumulada da receção total de uma stream, para o cenário de 400 nós



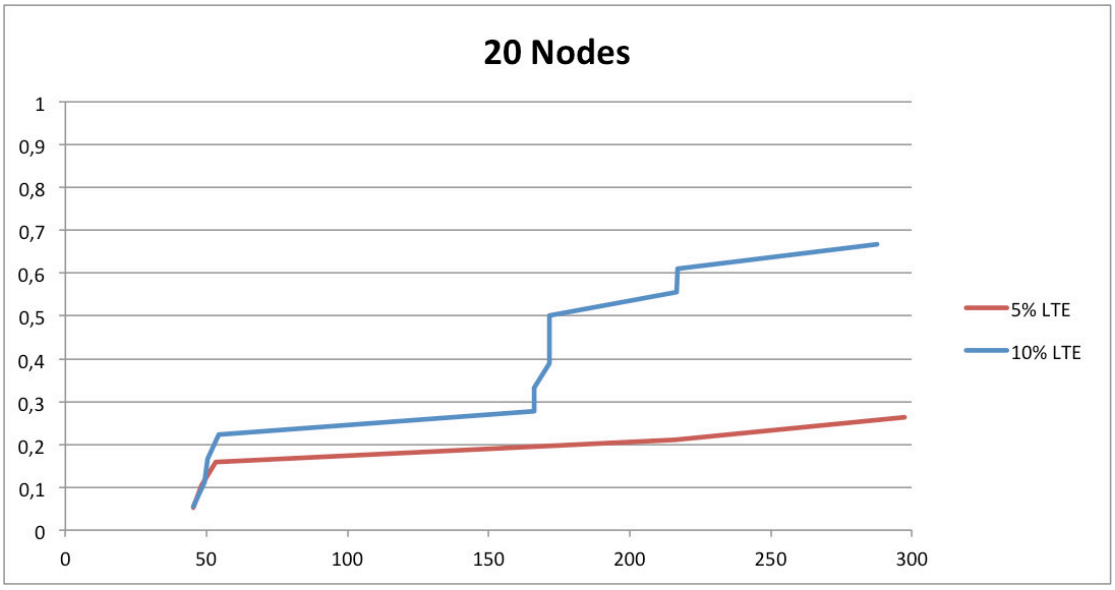
Anexo 6 - Função de distribuição acumulada da receção total de uma stream, para o cenário de 80 nós



Anexo 7 - Função de distribuição acumulada da receção total de uma stream, para o cenário de 40 nós



Anexo 8 - Função de distribuição acumulada da receção total de uma stream, para o cenário de 20 nós



Anexo 9 – Artigo submetido à WFCS 2015

Data Dissemination by Extending Publish/Subscribe to Vehicular Environments

Tiago Cerqueira, Michele Albano, Luis Lino Ferreira
CISTER Research Center
Polytechnic Institute of Porto (ISEP-IPP)
Rua Dr. António Bernardino de Almeida, 431
4249-015 Porto, Portugal
E-mail: {tcerq, mialb, llf}@isep.ipp.pt

***Abstract*—Vehicular communication technologies can provide valuable services to the drivers, providing them real-time information regarding the area where they are located. Data dissemination algorithms can earn benefits by employing different communication technologies at once, hence a vehicle may receive data from the infrastructure and from co-located vehicles that act as relay units. An early evaluation of the algorithms is usually done by simulation, but current vehicular simulation platforms are either using mobility models that are plainly unrealistic, or use traffic simulation systems that are complex to configure, thus leading to the necessity of expertise in an area that is outside usual ICT researchers competence.**

The research work described in this paper, concerns the design and implementation of a simulation platform providing realistic LTE, WAVE and mobility models, and simulations to evaluate a data distribution schemes in the Oporto area. Results show that exploiting broadcast communication of WAVE communication is a necessity to perform effective data distribution, and provide an estimation of needed vehicle density.

Keywords—ns-3; WAVE; LTE; mobility model

I. INTRODUCTION

The optimization of car's usage by means of Information and Communication Technology (ICT) showed strong potential for saving transportation energy, limit pollution, and reduce drivers' risks [1]. In the last few years, ICT usage to improve car's usage became a priority for the research agenda of the European Union (EU) [2], and a number of EU research projects [3] were funded to attain different goal in this area. The most general communication goals pertain to the production of platforms for efficient data collection and dissemination [4]. The latter objective provides the motivation for the research work described in this paper, which studies how to orchestrate ad-hoc and infrastructure communication to produce an efficient system to disseminate data to all the cars in an area.

Data dissemination to mobile nodes via wireless technologies is a problem that has been studied for a long time. In the case of vehicular scenarios, the academia and the industry agree for needing "the development of evaluation tools that define unified scenarios, and incorporate vehicular traffic patterns and channel models so that the merits and tradeoffs of the proposed protocols can be compared." [5].

The particular solution that is presented in this paper considers implementing a publish/subscribe system for data dissemination in vehicular networks, by orchestrating together communication through an infrastructure technology, and an ad-hoc technology. Example applications responding to this scenario comprise provisioning of traffic information in the neighborhood, advertising regarding the restaurants in the area, updating GPS maps with the latest data regarding road works in the area. All of these applications have the common characteristic of the data being valid in a limited geographical area only, thus justifying the usage of ad-hoc communication to propagate data in the neighborhood of already informed vehicles. The final goal of the investigation at hand aims at producing a solution for data dissemination that minimizes communication delay, while granting high data rate.

The evaluation of the dissemination algorithms implied a choice regarding the communication technologies to be simulated, and the decision was driven by the maturity of the technology and its potential for mass utilization in vehicular scenarios. As will be discussed in Section II, the Wireless Access in Vehicular Environments (WAVE) standard family is the main ad-hoc technology for Intelligent Transportation Systems (ITS) scenarios. In this context, WAVE is used by On-Board Units (OBUs) and Road-Side Units (RSUs) alike to implement both Vehicle to Vehicle (V2V) and Vehicle to Infrastructure (V2I) communication. The most promising candidate for infrastructure communication is LTE-Advanced (LTE-A) [6].

The rest of the paper is structured as follows. Section II describes related work. The resulting architecture for the system at hand is presented in Section III. The proposed dissemination protocol is described in Section IV. As it is common for novel protocols, the first analysis for the proposed approach is performed by means of simulations. The simulation platform built for this goal is based on network

simulator 3 (ns-3) [7], which is the most prominent tool used for both infrastructure and ad-hoc network simulation, and is described in Section V. The parameter space analyzed in the simulations, together with the results of the analysis is presented in Section VI. Finally, Section VII wraps up the paper and discusses future directions for the research work.

II. Related work

This section discusses the technologies that can be used as ad-hoc and infrastructure communication in the vehicular system we propose, simulation tools usually applied to vehicular scenarios, and protocols similar to the one proposed in this work.

Regarding ad-hoc communication, the Wireless Access in Vehicular Environments (WAVE) standard family is the main candidate protocol for the Intelligent Transportation Systems (ITS) scenarios. The protocol stack is articulated into the IEEE 802.11p protocol [8], which takes care of the PHY and MAC communication layers, and the IEEE 1609 family of standards [9] for the higher layers. WAVE is used by On-Board Units (OBUs) and Road-Side Units (RSUs) alike to implement both Vehicle to Vehicle (V2V) and Vehicle to Infrastructure (V2I) communication.

The IEEE 802.11p protocol is part of the IEEE 802.11 family, and it is based on the IEEE 802.11a [10] protocol, but optimized for V2V and V2I communication. The protocols share mechanisms such as Carrier Sense Multiple Access with Collision Avoidance (CSMA/CA) to arbitrate the access to the communication medium, Arbitration Inter-Frame Space (AIFS) to coordinate the access to the channels, and the Contention Window (CW) mechanism of IEEE 802.11e to grant Quality of Service (QoS) to communication. This latter technique provides, for both IEEE 802.11p and IEEE 802.11a, four different communication modes, to allow the coexistence of communications having different priority levels and requirements. Still, in the case of IEEE 802.11a the objective is to enable multimedia communication, while in IEEE 802.11p the objective for high priority communication is to support safety applications.

The main difference between IEEE 802.11p and IEEE 802.11a is that the data bandwidth of the former is much lower than the one of the latter, since IEEE 802.11p employs encoding mechanisms intended to make it more robust against the typical problems of ad-hoc networks, such as the hidden node (two nodes that are far away and try to communicate with a node between them at the same time, could be in trouble since they cannot sense each other while performing CSMA), and the high mobility of the nodes causing ever changing noise levels in the communication between them.

Regarding the higher layers, WAVE uses two different protocols as network layers:

- IPv6, which brings along the usual layers used in internet communication, such as TCP and RTP, and the mechanisms of IP protocol stack to protect and optimize the communication. IPv6 is the best candidate for transmitting usual internet data.
- Wave Short Message Protocol (WSMP), which does not employ a transport layer and is used for short and urgent messages. It relies on IEEE 802.11p mechanisms only to protect the data during transmission, and it has the advantage of being lightweight. WSMP is the best candidate for urgent and short messages.

The most promising candidate for infrastructure communication is LTE-Advanced (LTE-A) [6]. This protocol standard is based on LTE, which is oriented to human communication scenario, but it has a strong orientation to data communication, since latest releases (R-11 and R-12) of LTE-A present mechanisms to optimize M2M communication, starting with the improvement of the Radio Access Network (RAN) presented in TR 37.868 [11] and TR 23.888 [12]. Moreover, one of the profiles of LTE-A is tailored over the Vehicle-to-Infrastructure (V2I) communication scenario. Moreover, latest releases of the protocol are reducing medium access delays to provide support to soft real-time communication, and thus to safety and vehicular scenarios. The current Round Trip Time (RTT) of LTE-A is 20 ms, and in the next few years it is predicted to get as low as 5 ms.

Vehicular applications are currently relying to simulations for an early analysis before the deployment in real scenarios. This approach created the need for the industry and the academia alike, for simulation tools that can grant realistic results when challenged with problems characterized by high degree of complexity. In fact, in a vehicular scenario, multiple communication technologies can be used to enable V2V and V2I interaction, and the mobility models must be realistic for the movement of vehicles. The survey [13] analyses current simulation platforms, and discuss their pros and cons.

Currently, mobility models used for vehicular communication simulation are based either on basic trajectories (randomwalk, random waypoint model) [14] that do not take into account the map where the cars are moving, or resort to complex vehicular traffic simulations (for example, using SUMO, TraNS ou Bonnmotion) [15] to produce realistic synthetic traces. In this paper, we will consider using an external service to download travel planning information, and distil it into vehicular traces for the vehicles at hand. The resulting mobility, simulated by the RoutesMobilityModel module [16] of ns-3, proves to be quite realistic, while its inclusion in simulations being easily configurable and not requiring expertise in the vehicular traffic research area.

Previous works regarding data dissemination in vehicular environments include [17][18][19][20]. The work in [17] discusses opportunistic communication, considers that vehicles are willing to cache data they are not interested into, and uses a realistic modeling of Atlanta streets done by painful development work. The work in [18] uses the knowledge on the kind of the road the vehicle is on (important avenue vs minor street) to drive the data dissemination process, thus obliging the vehicle to acquire this information in some way. The protocol in [19] is similar to our one, but considers data that can be transmitted using just one message, and uses a complex traffic simulator to set up the nodes' routes. The work in [20] extends Delay Tolerant Networks (DTN)

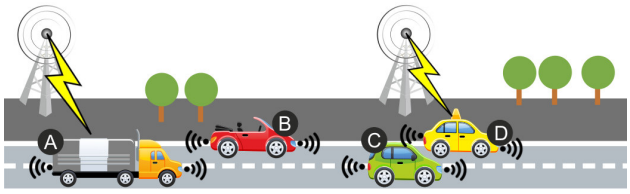


Figure 1 "Infrastructure plus ad-hoc" scenario

concepts to the vehicular scenario, uses a utility function to maximize the effect of each single message relay between the vehicles, and demonstrates the principles by a test-bed composed by just 2 vehicles. Moreover, all described works consider vehicles equipped with just one ad-hoc communication technology, and do not consider how many copies of the initial data are deployed into the network. In our case, we consider that each car is equipped with multiple technologies, vehicles are "selfish" (do not store data they are not interested into), the algorithm do not need knowledge regarding road network, and the mobility model is realistic but easily built through the use of our RoutesMobilityModel module [16].

III. System Architecture

This work considers that the distributed system performing data dissemination is composed by vehicles, an access network, and applications located in the internet. Each vehicle is equipped with an On-Board Unit (OBU) and network interfaces, plus other subsystems (sensors, screens, gps) that are not further specified since the proposed algorithm is agnostic to the disseminated data and thus is not concerned with the subsystems that make use of the received data.

The OBU has got a processing unit that can drive the execution of the algorithm, and enough memory to cache a large number of messages. Each vehicle is equipped with an ad-hoc interface, to allow them to communicate with each other, and exchange messages containing data; thus, the vehicles that cached data can also relay the data to other vehicles. Some of the vehicles are also equipped with an infrastructure wireless interface for communication with the access network. The access network is the entrance point to the internet, and can actually initiate communication with the vehicles equipped with an infrastructure wireless interface. The application located in the internet produce the data to be disseminated to the vehicles, and can broadcast the data to the access network, which thereafter contact vehicles in range to provide them the data.

IV. Data Dissemination in Vehicular Environments

This section discusses the extension of the publish / subscribe paradigm to vehicular networks, by orchestrating together communication through an infrastructure technology, and an ad-hoc technology. The resulting solution must be able to dissemination data with high data rate while minimizing communication delay. By orchestrating together long-range infrastructure communication and short-range ad-hoc protocols, and by adding a cache to vehicles to allow them to act as both recipients of the communication and data relay, the data bandwidth gets higher, and the communication is extended to vehicles not equipped with the long-range communication technology. Figure 1 provides a graphical representation of this scenario. We consider that data is produced on the internet. Should data be collected from the nodes themselves, or produced based on collected data, another protocol must be preliminarily run for data collection, but that is out of the scope of this paper.

Data sources are organized into N streams, the whole data produced by each stream is divided into a number M_N of chunks of the same size, and each chunk has got an expiration time. In the first phase of the algorithm the long-range base stations advertise their streams through the direct connection with registered nodes. Vehicles interested into the stream, and in range of the base stations, subscribe it and receive related chunks through usual internet technologies over the long-range communication technology. The vehicles maintain a cache with all the chunks they receive until their expiration time.

Later on, each node advertises the chunks it received through the ad-hoc communication technology by transmitting beacons. The beacon is used as both advertisement and keepalive message between the vehicles, and the payload of the beacon is the advertisement regarding streams active in the system, and chunks cached into the vehicle's cache. The structure of the beacon is as follows:

```
Advertisement ::=
                    #streams : integer
                    stream1 : Stream
                    ...
                    streamN : Stream;

Stream ::=
          name : String
          optionalFieldsStructure : byte
          optionalFields : various
          #chunks : integer
          chunk1 : Chunk
          ...
          chunkN : Chunk;

Chunk ::=
          unique_id : String
          expiry_time: DateTime;
```

The Stream structure comprises an `optionalFieldsStructure` bit mask, which defines the structure of the successive field `optionalFields`, which can contains Quality of Service requirements, priority levels, etc. When `optionalFieldsStructure` has its default value of 0, as is the case in the simulations we performed, the `optionalFields` is void and has a size of 0 bytes.

Each chunk reports the expiry date for the chunk itself, to allow both the cache manager to remove it when not valid anymore, and other vehicles to decide whether to download it or not depending on for how long it is going to be valid.

Communication via ad-hoc technologies extends the publish/subscribe mechanism through periodic transmission of

the subscribe message. The resulting schema is closer to a request/response approach. Vehicles communicate the stream they want to receive, and the recipients of the requests verify if they are subscribed to the requested stream and if they have relevant chunks to send. In this latter case, the recipients send the chunks regarding given streams back to the requesters. In the resulting schema, if the chunks are broadcast through the wireless medium, it is possible that vehicles receive chunks even when they do not requested them; on the other hand, vehicles can receive chunks they are not interesting into, thus creating the need for verification of the received chunks.

Therefore, the provided solution considers extending a publish/subscribe (push) mechanism through request/response (pull) interaction. The motivation is that the ad-hoc connections between vehicles concur to creating a communication environments characterized by intermittent communication, high bit error rates, and high churns in the connections between nodes, thus not granting the high reliability required by a pure publish/subscribe paradigm.

Simulations described in the following sections are aimed at evaluating, over different paths, the effect of having multiple streams against just one stream, the percentage of vehicles equipped with an infrastructure communication interface, the density of the vehicles, and the exploitation of the broadcast nature of the wireless ad-hoc communication. In fact, while the presence beacons that contain the requests are sent in broadcast, the chunks can be transmitted either through point-to-point connections between vehicles, or by broadcasting messages. The two solutions have different advantages when compared with each other:

- Unicast communication relies on sending at least one message for each message to be received. The resulting lower data bandwidth is paired with a higher flexibility, since it is possible to protect each connection with both acknowledgment mechanisms and security mechanisms for data confidentiality.

- Broadcast communication is able to disseminate data using a lower number of messages, since each message can be potentially received by many nodes, but on the other hand acknowledgment algorithms are very complex and cumbersome. Moreover, since all the interested nodes must be able to decipher incoming messages, security mechanisms must rely on key dissemination algorithms to share a session key, leading to more complex protocols.

Broadcast communication can reach more recipients and thus has higher performance, but it comes at the expense of security and robustness of the communication. Thus, Section VI draws a quantitative comparison to allow a system architect to perform an informed choice with respect to this trade-off.

V. Simulation platform

With the goals of simulating the protocols described in Section IV and providing support to further studies regarding vehicular communication, a simulator platform has been designed and implemented. The platform is based on network simulator 3 (ns-3) [7], which is considered the state-of-the-art for network simulation for both wired, wireless infrastructure and wireless ad-hoc communication. The simulator is event-based, provides output as both user-configured ASCII traces and pcap traces [7], supports the simulation of a large number of communication nodes, and allows the definition of mobility pattern of the nodes by importing traces that describe the trajectories taken by the nodes – in this case the vehicles.

The simulator uses several modules to access the functionalities needed for the analysis. The simulator is using modules for LTE protocol [21], WAVE protocol [22], and RoutesMobilityModel module [16] to create realistic mobility traces for the nodes.

The LTE module [21] is very mature, already provides all the functionalities needed for the analysis, and its only limitation is that it cannot simulate vertical handovers and roaming. Anyway, in the scenario at hand we are not interested in considering the latter effects, thus we did not have to take any actions on the LTE module.

The WAVE module [22] suffers from a lower level of maturity, and it has been included into the ns-3 distribution in January 2015. Current implementation includes PHY and MAC layers (IEEE 802.11p), but the implementation of IEEE 1609 (upper layers) is partial and it does not have a clear timeline for its completion. The research work produced a revision of current code, in particular regarding the Wave Short Message Protocol (WSMP), which is the network layer used in the scenarios at hand.

The RoutesMobilityModel [16], developed during the research work described in the paper, accesses an external service for trip planning, in this case the Google Maps Service [23], to download an XML file describing the travel planned for the vehicles, and parses it into the trajectories of communication nodes. The source and destination for a vehicle can be specified using an address, or through the Places API, thus easily generating mobility traces for every node. The mobility traces provided are based on real-world locations, which take into account the path a node would travel in the real-world trajectory and the speed the node would travel for the different road configurations (regular roads, highways, roundabouts, etc). The module works either with user specifying start and destination points, or with data obtained from the Google Maps Places API, to generate start and end points for multiple vehicles within a radius of a given real-world location with almost no user intervention.

VI. Simulation results

The simulated scenarios compare the execution of the algorithm on different trajectories of the vehicle, different number of streams in the system, different percentage of vehicles equipped with LTE interface, and different transmission modes for the chunks. The result is an evaluation of the algorithm in terms of requirements regarding the minimum number of vehicles that must receive the data through the long-range technology, required density of communicating vehicles, and the effect of communicating the chunks via unicast communication. Table 1 resumes the parameters spanning the problem space we analyzed. All simulations feature a total of 240 vehicles, the total simulated time is 1200 seconds (20 minutes), and all vehicles are equipped with a WAVE interface. The ns-3 version used in the

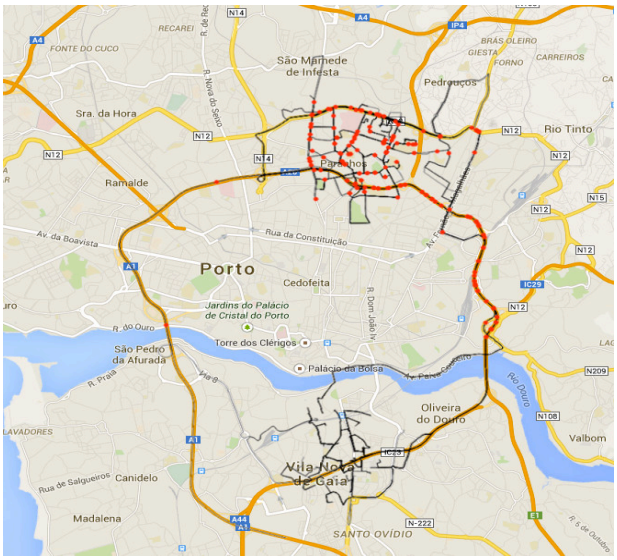


Figure 2 Path between CISTER's and Gaia

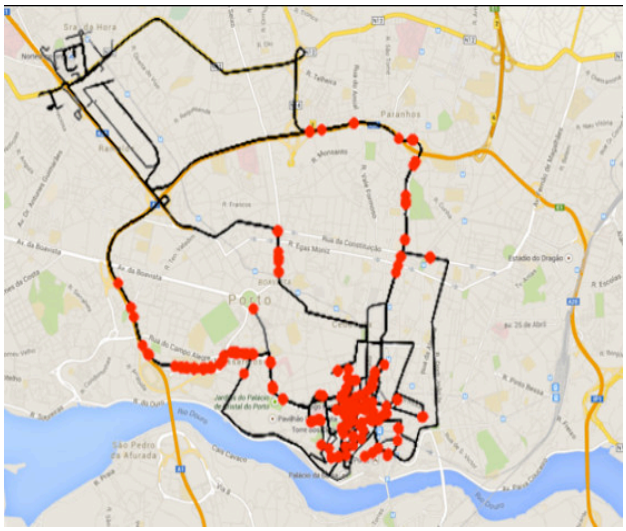


Figure 3 Path between downtown Oporto and Norte Shopping

paper is 3.19, and the WSMP routing layer was employed for WAVE communication.

The parameter Path describes which mobility scenario was used in the simulation. Scenario CG defines that the vehicles are initially positioned in a circular area of 1000m radius around CISTER Research Center in the north part of Oporto, and the destinations of the paths are in a circular area of 1000m radius in Gaia, which is located just south of Oporto (Figure 2). Scenario DN deploys the vehicles in a circular area of 500m in Downtown Oporto, and states that cars' destinations are in a circular area of 500m around Norte Shopping, a large commercial center located in the industrial zone of Oporto (Figure 3). Scenario DND, also represented in Figure 3), sets initially 50% of the cars in Downtown Oporto, with destination Norte Shopping, and 50% of the cars in the Norte Shopping area, with destination Downtown Oporto. In all the considered cases, the exact initial position of each car is decided through the module RoutesMobilityModel [16], which accesses the Google Places service [23] to choose locations in the area defined in each scenario. Thus, vehicles cars in Downtown Oporto will actually be at different address in the downtown area. Resulting initial vehicle densities are reported in Table 2.

Table 1 Simulation parameters

Path	CG, DN, DND	The start / destination of the vehicles
LTE	5%,10%,20%, 30%,40%,50 %	Percentage of cars equipped with the long-range communication technology
Send_mode	Broadcast, unicast	Transmission mode for the chunks
Streams	1,2	Number of streams in the system

The LTE parameter defines the percentage of cars that are equipped with an LTE interface (all the cars are equipped with the WAVE interface).

The parameter Send_mode is related to the transmission mode for the chunks. The beacons are always sent in broadcast, but the chunks are sent either in unicast, to simulate a point-to-point transmission through a protected channel, or in broadcast, to allow all co-located cars to receive the chunk. The two communication modes respond to different functional / non-functional requirements:

- With the unicast communication mode, secure / safe communication is easier to implement, since it is sufficient to add acknowledgment mechanisms and channel encryption, at the cost of lower performance;
- In the broadcast case, the performance gets better since all vehicles will be able to receive data every time that a transmission occupies the wireless medium. On the other hand, more complexity must be coped with to make the communication robust (accumulated acknowledgments? statistical acknowledgments?) and secure (shared group keys? session keys?).

The parameter Streams indicates if all the data in the area areis associated to one sStream, or the system is characterized by 2 different streams. In the first case, all vehicles subscribe to the stream. In the case of the 2 streams, each car vehicles has got a 70% chance probability of subscribing each stream, thus 9% of the cars do not subscribe any stream, 49% both the streams, 42% either one of the streams.

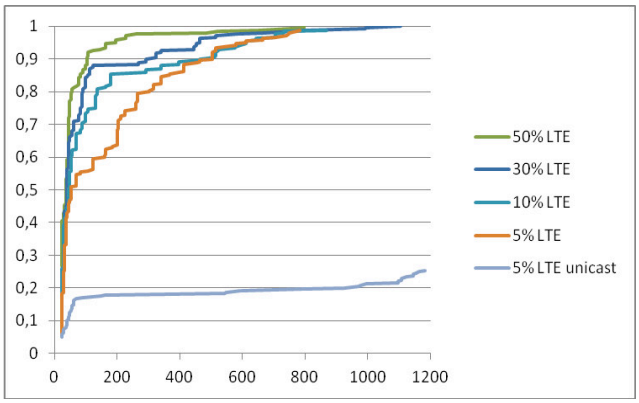


Figure 4 CISTER to Gaia, 1 stream

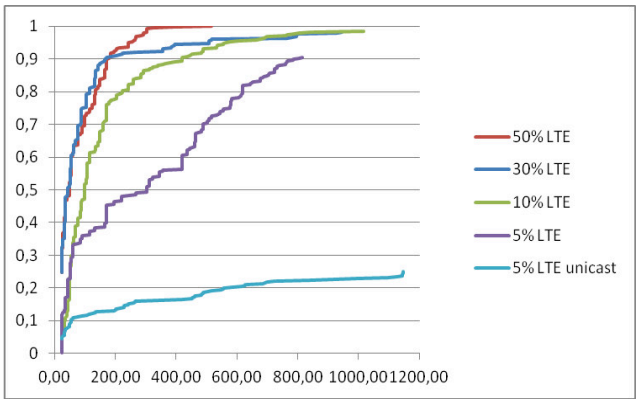


Figure 5 CISTER to Gaia, 2 streams

The simulations whose results are represented in Figure 4, are focused onto the scenario where just 1 stream is present in the system (Streams=1) and the vehicles go from CISTER's area to Gaia (Path=CG). The percentage of vehicles equipped with LTE interfaces varies (LTE=5%, 10%, 30%, 50%), and communication mode is always in broadcast (Send_mode=broadcast) except in one simulation where the investigation was focused on the effect of unicast communication (LTE=5%, Send_mode=unicast). The simulated time was 20 minutes, which is more than enough to bring all the cars to their destinations. The graph represents the fraction of vehicles that received all required data streams, against simulation time (cumulative distribution function, or CDF, of completed stream transmissions vs elapsed time). As expected, the effect of unicast communication is much more dramatic than reducing the percentage of LTE-equipped vehicles, in fact in all broadcast scenarios, after a reasonable time, close to all the vehicles received all the streams. On the other hand, in the case of unicast communication, only 25% of the vehicles received required data, and just 20% in a reasonable time (before getting to the destination of the vehicle). A larger fraction of LTE-equipped vehicles provided clear speed-up to the algorithm, but as long as enough data copies get into the distributed system through the long-range technology, ad-hoc communication will take care of the epidemic delivery of the data.

The simulations related to Figure 5 extend the previous reasoning to a system where 2 streams are present, and each stream is subscribed by each vehicle with a probability of 70% (Streams=2). The rest of the parameters are set in the same way of the previous case. Unicast communication is still a clear loser to broadcast communication. The only sensible difference with previous case is that broadcast communication for the ad-hoc technology is not sufficient to perform a proper dissemination of the data, when 5% of the vehicles are equipped with a LTE interface. In fact, while in the previous case a mean of 12 vehicles subscribed the stream via LTE technology, this time each stream gets subscribed by a mean of 8 vehicles. The behaviour evidenced in the graph is that such a small number of initial data does not allow the epidemic dissemination to complete data distribution to all the vehicles. Still, just 10% of the subscribers do not receive the data, thus the system performance is sufficient for some applications (advertising?), while other applications would behave in an unacceptable manner (safety-related applications).

Last scenarios analyzed in our simulations concern the path from Oporto downtown to Norte Shopping (Path=DN), and the case where half the vehicles were initially in downtown Oporto directed to Norte Shopping and the rest were doing the opposite trajectory (Path=DND). All data are part of the same stream (Streams=1), and 5% of the vehicles are equipped with LTE interfaces (LTE=5%). Simulations were performed for both unicast and broadcast WAVE chunk communication. As shown in Figure 6, in the Path=DN case, there are less alternative roads between start and destination, and the density of the vehicles is 305 instead of 76 of CG case (see Table 2), thus the vehicles get in contact more often, leading to a higher performance of ad-hoc data distribution. Still, the performance of unicast WAVE transmission proved to be undesirable. In the case of DND path, the performance gets lower, since the density is 152 (see Table 2) in each initial area. The number of contacts between vehicles in each of the two areas, which grows like the second power of vehicle density, leads to a total number of contacts that is approximately 50% than in the DN case, and to the lower performance of data dissemination for the DND scenario.

VII. Conclusions

The research work described in this paper, built a simulation platform to analyze data dissemination algorithms involving vehicles equipped with ad-hoc and infrastructure wireless interfaces. The work considered implementation of the communication through WAVE and LTE protocols, it estimated the required density of WAVE and LTE vehicles in two real-life scenarios, and compared different ways of distributing data. Unicast communication for WAVE technologies proved to have an unacceptable performance for the algorithm at hand, while exploiting broadcast access to the wireless medium allowed WAVE communication to propagate

Table 2 Vehicle density per scenario

CG	76 Vehicles / km^2
DN	306 Vehicles / km^2
DND	153 Vehicles / km^2

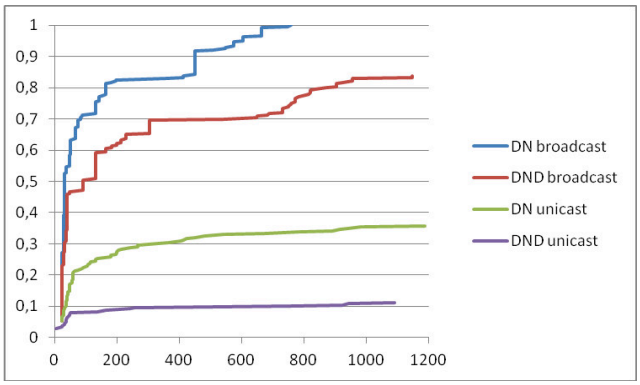


Figure 6 Different paths, 1 Stream

data in a more effective manner.

Future work will compare the proposed algorithm with state-of-the-art approaches, thus harvesting further the advantages of the simulation platform (WAVE protocol simulation, LTE protocol simulation, realistic mobility model).

ACKNOWLEDGMENTS

This work was supported by The Portuguese Agency for Innovation (ADI) under the ERDF (European Regional Development Fund) through COMPETE (Operational Programme 'Thematic Factors of Competitiveness'), within project CARCODE, ITEA2 Nr. 11037, QREN - SI I&DT Nr. 30345.

REFERENCES

[1] Papadimitratos, Panagiotis, et al. "Secure vehicular communication systems: design and architecture." Communications Magazine, IEEE 46.11 (2008): 100-109.

[2] Cahill, Vinny, et al. "The managed motorway: real-time vehicle scheduling: a research agenda." Proceedings of the 9th workshop on Mobile computing systems and applications. ACM, 2008.

[3] Car2Car Consortium. "CAR 2 CAR Related Projects", June 2014, available online at: https://www.car-2-car.org/index.php?id=6

[4] Al-Sultan, Saif, et al. "A comprehensive survey on vehicular Ad Hoc network." Journal of network and computer applications 37 (2014): 380-392.

[5] Chen, Wai, et al. "A survey and challenges in routing and data dissemination in vehicular ad hoc networks." Wireless Communications and Mobile Computing 11.7 (2011): 787-795.

[6] Ghosh, Amitava, et al. "LTE-advanced: next-generation wireless broadband technology [Invited Paper]." Wireless Communications, IEEE 17.3 (2010): 10-22.

[7] Henderson, Thomas R., et al. "Network simulations with the ns-3 simulator." SIGCOMM demonstration (2008).

[8] Jiang, Daniel, and Luca Delgrossi. "IEEE 802.11 p: Towards an international standard for wireless access in vehicular environments." Vehicular Technology Conference, 2008. VTC Spring 2008. IEEE. IEEE, 2008.

[9] Hartenstein, Hannes, and Kenneth P. Laberteaux. "A tutorial survey on vehicular ad hoc networks." Communications Magazine, IEEE 46.6 (2008): 164-171.

[10] Meng, Teresa H., et al. "Design and Implementation of an All-CMOS 802.11 a Wireless LAN Chipset." IEEE Communications Magazine (2003): 161.

[11] 3GPP consortium, "Technical Report 37.868", available online at http://www.3gpp.org/ftp/specs/htmlinfo/37868.htm

[12] 3GPP consortium, "Technical Report 23.888", available online at http://www.3gpp.org/ftp/specs/htmlinfo/23888.htm

[13] Martinez, Francisco J., et al. "A survey and comparative study of simulators for vehicular ad hoc networks (VANETs)." Wireless Communications and Mobile Computing 11.7 (2011): 813-828.

[14] Camp, Tracy, Jeff Boleng, and Vanessa Davies. "A survey of mobility models for ad hoc network research." Wireless communications and mobile computing 2.5 (2002): 483-502.

[15] Martinez, Francisco J., et al. "A survey and comparative study of simulators for vehicular ad hoc networks (VANETs)." Wireless Communications and Mobile Computing 11.7 (2011): 813-828.

[16] T. Cerqueira, M. Albano, "RoutesMobilityModel: easy realistic mobility simulation using external information services", unpublished, submitted to Workshop on ns-3 (WNS3 2015)

[17] Wu, Hao, et al. "MDDV: a mobility-centric data dissemination algorithm for vehicular networks." Proceedings of the 1st ACM international workshop on Vehicular ad hoc networks. ACM, 2004.

[18] Zhao, Jing, Yang Zhang, and Guohong Cao. "Data pouring and buffering on the road: A new data dissemination paradigm for vehicular ad hoc networks." Vehicular Technology, IEEE Transactions on 56.6 (2007): 3266-3277.

[19] Ros, Francisco J., Pedro M. Ruiz, and Ivan Stojmenovic. "Acknowledgment-based broadcast protocol for reliable and efficient data dissemination in vehicular ad hoc networks." Mobile Computing, IEEE Transactions on 11.1 (2012): 33-46.

[20] Schwartz, Ramon S., et al. "On the applicability of fair and adaptive data dissemination in traffic information systems." Ad hoc networks 13 (2014): 428-443.

[21] Herman, Budiarto, et al. "Extensions to LTE mobility functions for ns-3." Proceedings of the 2014 Workshop on ns-3. ACM, 2014.

[22] Bu, Junling, et al. "Implementation and evaluation of WAVE 1609.4/802.11 p in ns-3." Proceedings of the 2014 Workshop on ns-3. ACM, 2014.

[23] Miller, Christopher C. "A beast in the field: The Google Maps mashup as GIS/2." Cartographica: The International Journal for Geographic Information and Geovisualization 41.3 (2006): 187-199.

Anexo 10 - Apresentação realizada na WNS3 2015, sobre o modelo de mobilidade desenvolvido

CISTER
Research Centre in
Real-Time Computing Systems

RoutesMobilityModel: Easy Realistic Mobility Simulation using External Information Services

Tiago Cerqueira, Michele Albano
CISTER Research Centre, ISEP, Portugal
www.cister.isep.ipp.pt

Workshop on Network Simulator 3, 2015
May 13th, 2015

isep Instituto Superior de Engenharia do Porto

Outline

- Mobility models for ns-3
  - ns3::RandomWaypointMobilityModel
  - SUMO
- Using external information services
  - Google Maps API
- ns3::RoutesMobilityModel
  - Features
  - Results
  - Internals
  - Limitations
- Future (current) work

Mobility Models for ns-3

- Simulation of mobile communication makes use of mobility models
- ns-3 possesses several synthetic mobility models implemented, such as:
  - Random Waypoint Mobility Model
  - Random Walk 2D
  - Gauss-Markov Mobility Model
- Another approach is coupling ns-3 with a traffic simulator, such as
  - Simulation of Urban MObility (SUMO)
  - Multi-agent Microscopic Traffic Simulator (MMTS)

ns3::RandomWaypointMobilityModel

- This module picks for each node a destination and velocity at random. When the node reaches the destination, it pauses for a specified amount of time and restarts the process.

- Easy to configure:
  - Only parameters: area where to place the nodes, speed range, pause time

- Disadvantages
  - Not much realism (constant speed, random positions, random destinations, does not consider the underlying road network)

SUMO

- SUMO is a microscopic vehicular traffic simulator
- Provides lots of interesting features
  - Can model vehicles, pedestrians and public transport
  - Can import maps, or generate custom road networks
  - Can model car following models and inner junction traffic
  - Gas emissions, traffic light frequency, etc
- Disadvantages
  - Steep learning curve
  - Can potentially take long time to configure since many details can/have to be specified
  - The user documentation consists mainly on its wiki
  - The SUMO architecture consists on a number of small programs that comprise the SUMO suite
    - Need to know, and possibly configure and run each tool to maximize realism

Using an external Information Service

- Synthetic mobility models in ns-3 are relatively simple and straightforward to use
  - However, they suffer from a low degree of realism.
- Full-fledged traffic simulators such as SUMO are more realistic
  - Time-consuming configuration
  - Easy to misconfigure / need expertise in vehicular traffic simulation
- We propose to distill data from travel planning services into mobility directives
  - We built a prototype that makes use of Google Maps API

Why Google Maps API?

- The Google Maps Web Services are one of the main core products of Google and provide geographic data for applications
  - Good support to developers
  - Feature rich

- We propose to use two APIs in particular:
  - Directions API
    - This API retrieves direction information between two locations
  - Places API
    - This API retrieves real world places (restaurants, hospitals, etc) around a given real world location

ns3::RoutesMobilityModel

- What it is
  - An interface to request directions between two (or more) points from external services, and parse them into a ns-3 mobility
  - A trade off between complex traffic simulations and easy (unrealistic) synthetic mobility trace generators
- What it isn't
  - A full-fledged traffic simulator for ns-3

- Currently, implemented for Google Maps only
- Can function online (contacting external service) or offline (importing travel plans from filesystem)
- Allows to select transportation method (driving, walking, cycling and public transportation)
- Allows to specify a departure time
  - Especially important when modeling public transportation

Visual comparison

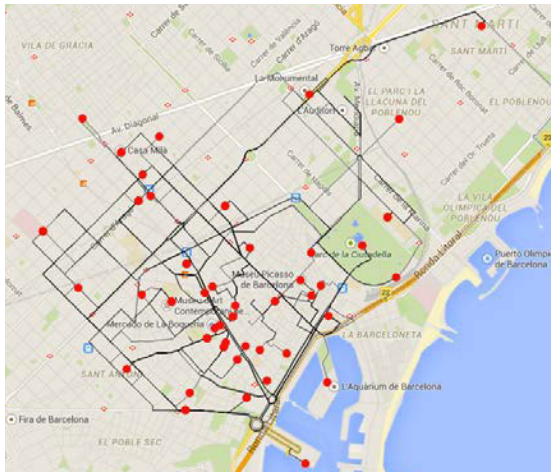


Figure 1 – RoutesMobilityModel

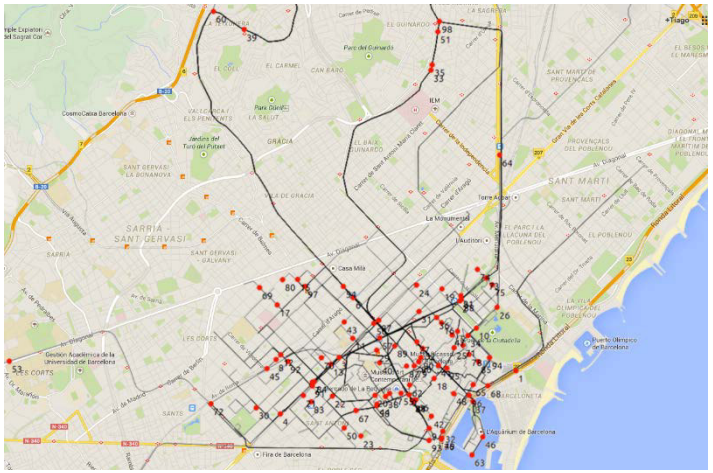


Figure 2 – SUMO

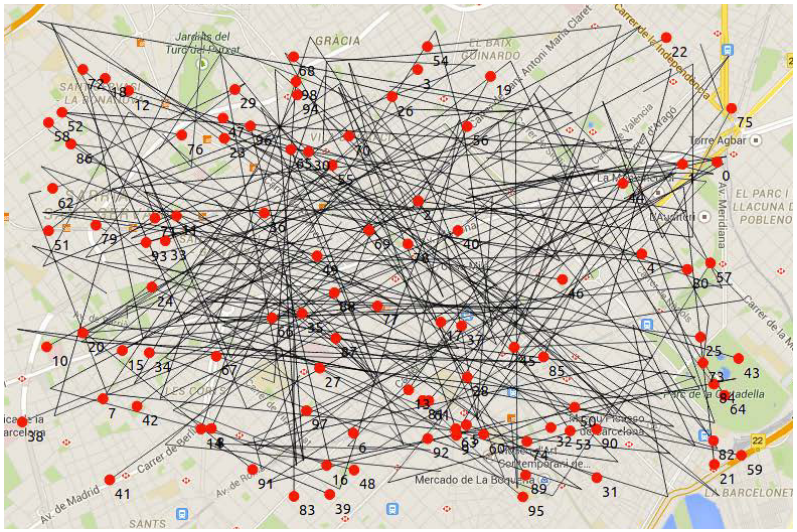


Figure 3 – RandomWaypointMobilityModel

Configuration complexity comparison

```
MobilityHelper mobility;
//Assign initial random positions
ObjectFactory pos;
int64_t streamIndex = 0;
pos.SetTypeId ("ns3::RandomBoxPositionAllocator");
pos.Set ("X", StringValue ("ns3::UniformRandomVariable[Min=0.0|Max=4600.0]"));
pos.Set ("Y", StringValue ("ns3::UniformRandomVariable[Min=0.0|Max=3000.0]"));
pos.Set ("Z", StringValue ("ns3::UniformRandomVariable[Min=1.0|Max=2.0]"));

Ptr<PositionAllocator> pAlloc = pos.Create ()->GetObject<PositionAllocator> ();
streamIndex += pAlloc->AssignStreams (streamIndex);
//Configure the speed and the pause time
std::stringstream ssSpeed;
ssSpeed << "ns3::UniformRandomVariable[Min=0.0|Max=" << 11.2 << "]";
std::stringstream ssPause;
ssPause << "ns3::ConstantRandomVariable[Constant=" << 5 << "]";
mobility.SetMobilityModel ("ns3::RandomWaypointMobilityModel",
                           "Speed", StringValue (ssSpeed.str ()),
                           "Pause", StringValue (ssPause.str ()),
                           "PositionAllocator", PointerValue (pAlloc));
mobility.SetPositionAllocator (pAlloc);
//Install the mobility to the nodes
mobility.Install (nodes);
streamIndex += mobility.AssignStreams (nodes, streamIndex);
```

Figure 4 – ns3::RandomWaypointMobilityModel configuration

```
MobilityHelper mobility;

mobility.SetMobilityModel ("ns3::WaypointMobilityModel");

mobility.Install(nodes);

RoutesMobilityHelper routes (41.306717, 2.119782,0);

routes.ChooseRoute(nodes,41.385329, 2.179434,2000);
```

Figure 5 – ns3::RoutesMobilityModel configuration

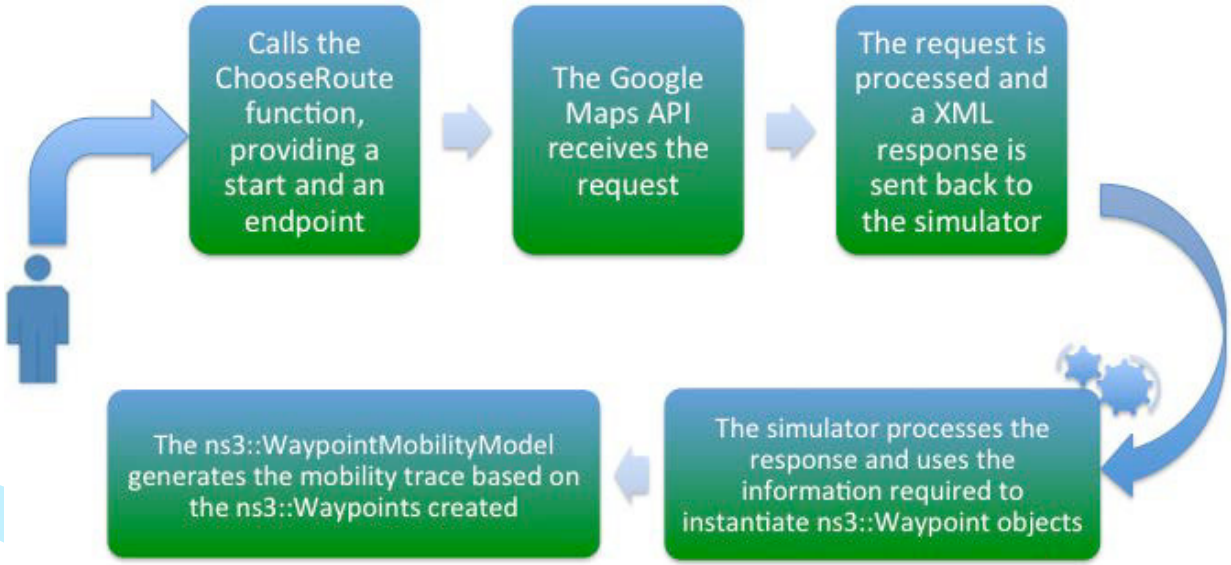
```
Export OSM map
netconvert --osm-files Downtown\ Barcelona.osm -o dwntwnBarcelona-net.xml
polyconvert --net-file dwntwnBarcelona-net.xml --osm-files Downtown\ Barcelona.osm --type-file typemap.xml -o DowntownBarcelona-poly.xml
python /usr/share/sumo/tools/trip/randomTrips.py -n dwntwnBarcelona-net.xml -r barcelona-rand.xml -e 50 –l
sumo -c config.sumo.cfg --fcd-output sumoTraceBarcelona.xml
python ../tools/traceExporter.py --orig-ids --fcd-input sumoTraceBarcelona.xml --ns2mobility-output mobility.tcl

Each line corresponds to an action, which has got its own configuration file
Some of the files (e.g. typemaps.xml, a basic one is 36 lines) MUST be filled up for each scenario
```

Figure 6 – SUMO configuration

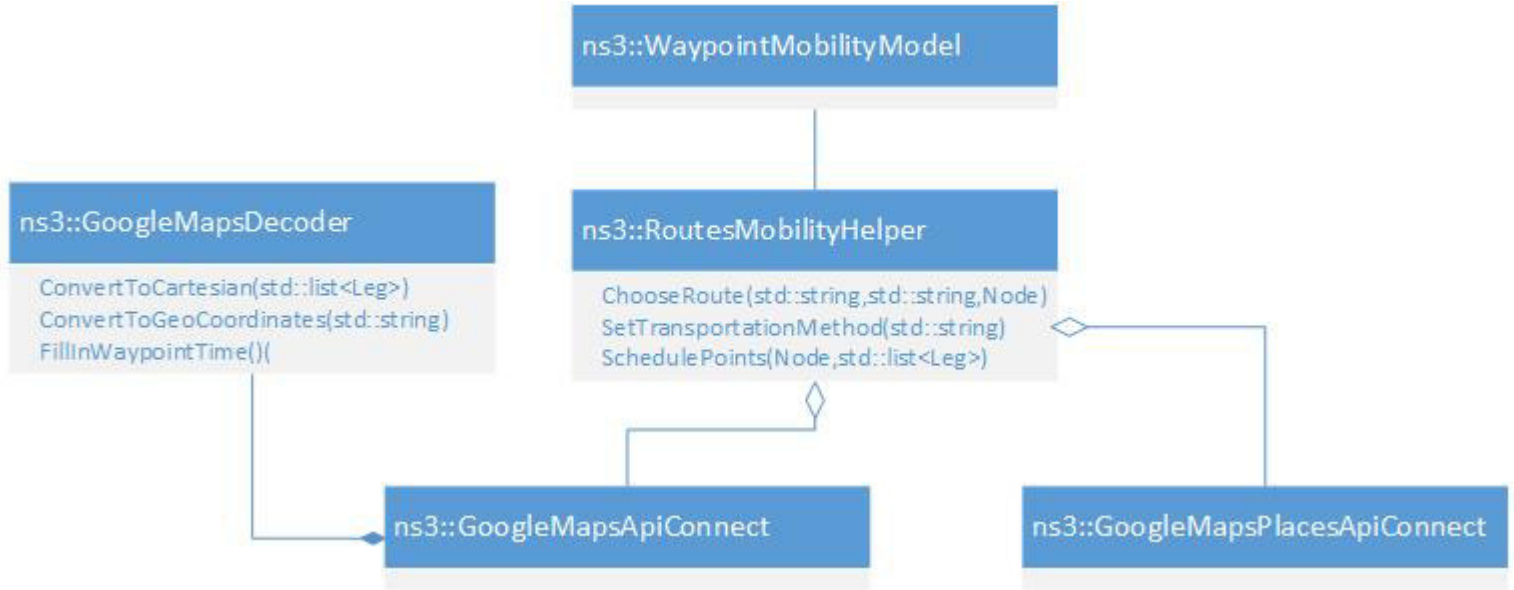
Internals 1/2

- The module was designed to import the routes via ns3::WaypointMobilityModel
  - Our module retrieves Geodetic points provided by Google Maps and converts them to Cartesian points, which can then be added to the WaypointMobilityModel



Internals 2/2

- The design of the module allows to download travel plans from different services, e.g. OSM
  - Module implemented using the Strategy Design Pattern, to adapt to additional external directions services
    - New strategy to access the service and parse the retrieved travel plan
  - Currently only the Google Maps services are being used



Limitations

- Current release of the module can/will be improved:
  - Only possible to generate mobility for node containers up to 30 nodes
    - Due to a limitation of the Places API
  - The module takes a long time to parse XML responses into mobility for large node containers
    - Still much faster than the execution of any non-trivial ns-3 simulation
  - No serialization is implemented
  - No bidirectional coupling with ns-3 is implemented
    - Currently, mobility impacts on what happens ICT-side
    - We want to allow ICT (car's computations) to lead to decision in changing car's routes
  - Traffic information is not available in the free version of the API

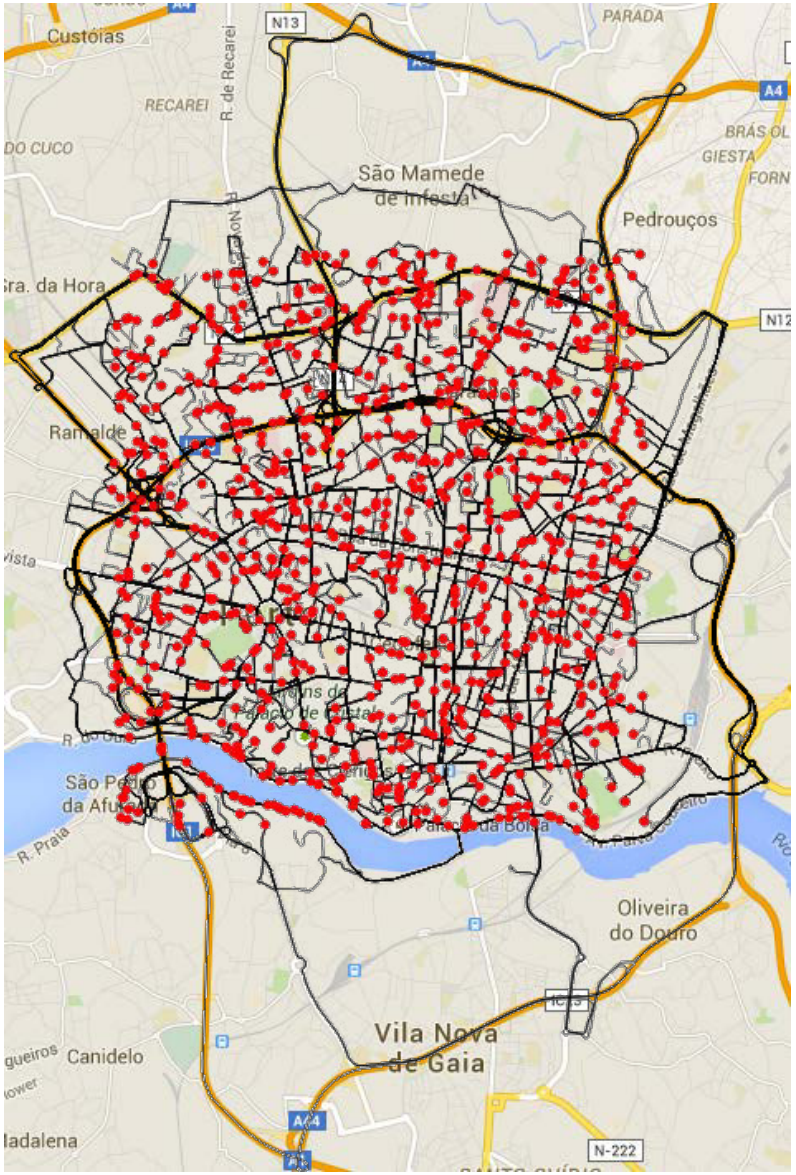
Future/current work

- Removing the 30 node limitation
- Switch from XML to JSON
  - Potential of speeding up the parsing of mobility data
- Implement serialization
  - To allow the user to repeat the simulation using the exact same mobility
- Bidirectional coupling with the simulator
- Thorough evaluation of the module
  - Other metrics
  - Other mobility models

Sneak preview of the next release

- Already available in the repository
  - NOT under review for ns-3 inclusion
  - Will not be included in ns-3's next version
    - Hopefully will be included later on
- The 30 node limitation was removed
  - By combining the 60 places returned by the Places API at random
  - By combining user specified locations at random
  - By randomly choose coordinates in a user specified location
- Serialization
  - In the current release if a user manually downloads a travel plan, the module can already parse it as mobility

Sneak preview of the next release



Thank you for your attention

- Current testing repository
  - https://bitbucket.org/TiagoCerqueira/routesmobilitymodel
- Wiki
  - https://www.nsnam.org/wiki/RoutesMobilityModel
- Code review issue
  - https://codereview.appspot.com/176430044/

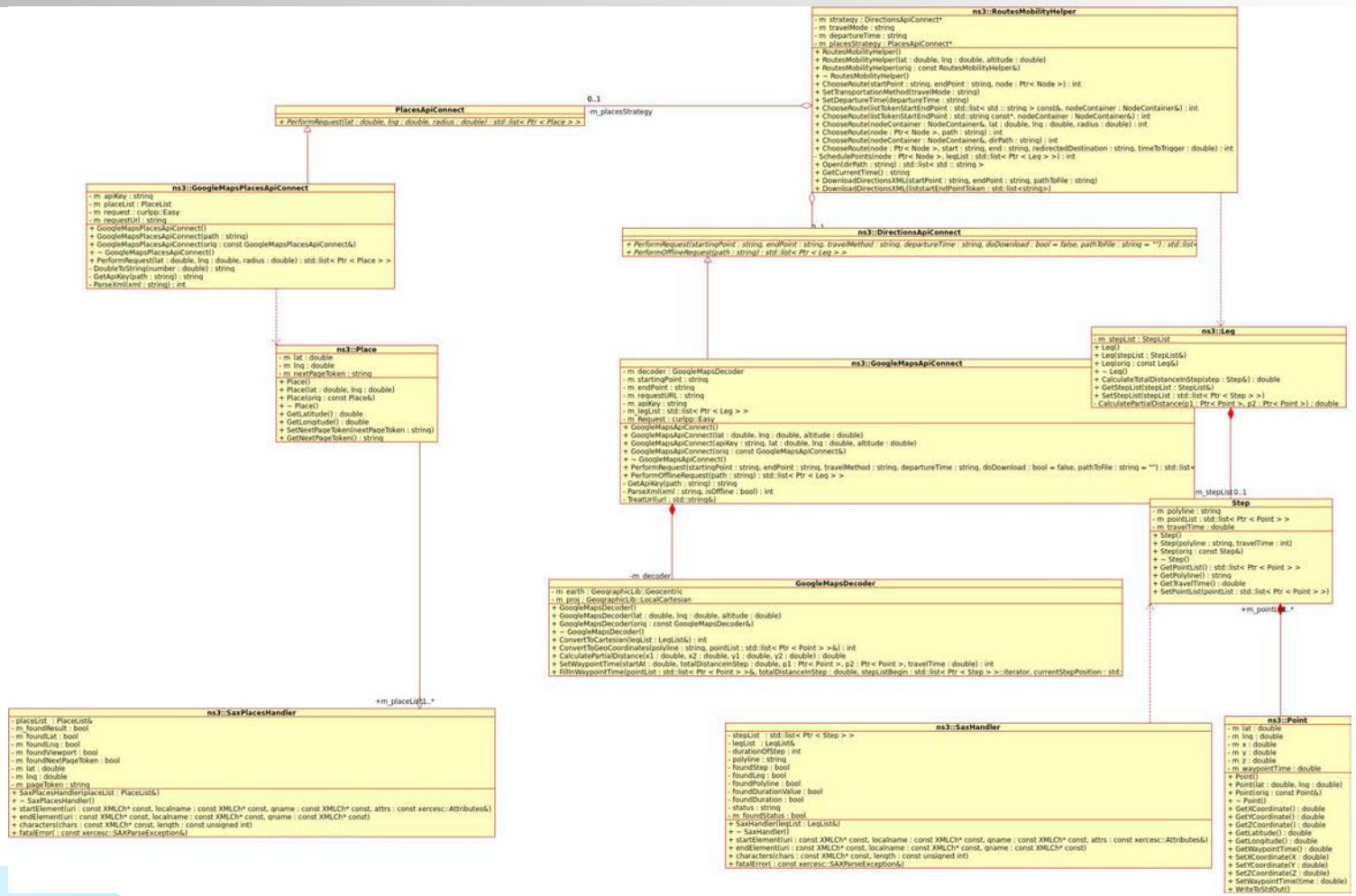
More material

- Both these APIs are free and have usage quotas
  - Directions API
    - 2.500 Directions requests per 24 hour period
    - 2 requests per second
    - No traffic information is available on the free version
  - Places API
    - 1000 requests per 24 hour period, which can be increased to 150 000 requests by verifying the user's identity with a credit card.
    - Unfortunately it's only able to return up to 60 places in any query (despite the fact that there are more locations in that area)

More material

- Caching the API's responses for more than 30 days is a direct violation of the ToS. Because of this, no caching feature was implemented. However, it is possible to load XML responses from the filesystem

More material



MacPhy overhead comparison

- An analysis of the performance of the routing protocol AODV was performed using vanet-routing-compare script, in order to further validate the results
  - Three scenarios where tested:
    - A scenario using the ns3::RoutesMobilityModel on Downtown Barcelona
    - A scenario using SUMO on Downtown Barcelona
    - A scenario using the ns3::RandomWaypointMobilityModel using the same area as the area in the scenarios using SUMO and the RoutesMobilityModel.
- The simulations where ran using 99 nodes, for 300 simulated seconds. The vehicles randomly chose the route to take.