Universidade de Aveiro

Gestão de Parques de Estacionamento
Redes e Sistemas Autónomos

João Gameiro (93097), Marco Ramos (93388)

Departamento de Electrónica, Telecomunicações e Informática

24 de junho de 2022

Conteúdo

Lista de Figuras

Capítulo 1

Introdução

O presente relatório visa descrever os passos e processos realizados no desenvolvimento no âmbito da unidade curricular de Redes e Sistemas Autónomos.

O código desenvolvido para o projeto encontra-se disponível em: https://github.com/JPCGameiro/RSA_Project.

1.1 Objectivo

O objetivo principal deste trabalho é desenvolver uma rede veicular que permita troca de informação do número de lugares de estacionamento disponíveis a cada momento. A ideia que deu origem a este projeto assenta no fundo numa troca de mensagens entre carros e unidades à entrada de parques de estacionamento, isto é um carro quando se aproximasse de um parque de estacionamento receberia uma mensagem da unidade à entrada do parque que iria conter a ocupação do parque. De acordo com a ocupação recebida, era possível tomar a decisão de seguir em frente à procura de outro lugar ou ir imediatamente para o parque em questão.

Esta ideia iria permitir assim uma maior fluidez de trânsito (evitar filas para parques de estacionamento) assim como auxiliaria na poupança de tempo por de parte dos condutores na estrada na medida em que não se perde tanto tempo à procura de lugar para estacionar por se saber com antecedência se existe disponibilidade no parque ou não.

No planeamento e desenvolvimento do projeto, identificaram-se duas entidades:

- **RSUs - Road Side infrastructure Units** - que são nós da rede equipados com módulos para processamento e comunicação wireless. No contexto do projeto seriam responsáveis por controlar que carros estão estacionados e enviar informação relativa à ocupação dos parques aos veículos que circulam na estrada quando necessário.
- **OBUs - On Board Units** - que são equipamentos para comunicação wireless colocados nos veículos que circulam que no contexto do projeto enviariam mensagens às ***RSUs*** a informar da sua presença na estrada

A partir dos dois tipos de entidades referidos em cima, identificam-se dois tipos de comunicação diferentes no contexto de redes *VANET* (Vehicular Ad hoc Networks):

- **Vehicle-to-Vehicle (V2V)** Comunicação direta entre veículos que contém informação como velocidade, direção e as suas respetivas características.
- **Vehicle-to-Infrastructure (V2I)** Comunicação direta entre os veículos e as infraestruturas da estrada, que no caso deste projeto é usada para reportar a ocupação dos parques.

1.2 Mensagens

Para comunicar entre os vários elementos da rede foi necessário usar mensagens em dois formatos específicos pré-definidos: ***CAMs*** e ***DENMs***.

1.2.1 CAMs

CAMs (Cooperative Awareness Messages) são mensagens usadas para informar outros elementos da estrada da presença de um dado veículo. Contém informação como por exemplo: a velocidade, posição do veículo, sistemas ativos (p.e.: estado dos pedais), direção, dimensão, tipo de veículo, etc. São enviadas periodicamente com uma frequência de 1Hz a 10Hz.

1.2.2 DENMs

DENMs (Decentralized environmental notification messages) são mensagens usadas para sinalizar outros nós da rede veicular sobre eventos presentes na estrada como por exemplo, um acidente ou engarrafamentos, etc. São geradas à medida que o evento ocorre, não sendo por isso periódicas e têm um tempo de validade. No entanto no caso deste projeto elas foram usadas com o propósito de notificar os carros da ocupação dos parques de estacionamento. Esta decisão foi tomada pelo facto de não existir nenhum tipo de mensagem adequado a esta situação especifica. Por isso para adequar a ***DENM*** à situação foi usado um novo *causeCode* com o intuito de representar um novo tipo de evento que é a ocupação do parque de estacionamento que é representada no campo *subCauseCode*.

1.3 Tecnologias e ferramentas

Para desenvolver esta rede veicular foi utilizada a extensão *NAP-Vanetza* que integra *MQTT* e *JSON*. Esta extensão permite implementar a comunicação com mensagens no formato *ETSI C-ITS* entre os vários nós da rede, lidando com processamento envio e receção destas mensagens.

Para construir a base de dados foi utilizada a biblioteca *SQLite3*, por se tratar de uma ferramenta simples e rápida de montar ideal para o projeto apresentado.

A linguagem de programação usada para integrar os módulos anteriores e criar a simualação foi *Python*.

Para a apresentação gráfica da simulação foi desenvolvida uma aplicação web com bibliotecas que suportam mapas interactivos. O mapa foi construido com recurso à biblioteca *Javascript, Leaflet*. Foi também usado *HTML* e *CSS* para servirem como base ao mapa apresentado. A tecnologia para o servidor foi *Flask*.

1.4 Metedologia Seguida

O primeiro passo realizado para implementar este projeto, foi o de estudar o sistema *vanetza* assim como *mqtt* para perceber como lidar com toda a parte de comunicação entre nós da rede. Paralelamente a este processo foram estudados os formatos das *CAMs* e da *DENMs*.

Após ter percebido o formato ideal e como funciona o *vanetza*, foi realizada a discussão sobre como criar e adicionar mais nós à simulação, assim como a criação de *scripts* que gerassem de forma automática as mensagens usadas na comunicação entre elementos da rede. Surgiu assim a primeira simulação simples com apenas duas *OBUs* e uma *RSU* com um parque a si associado.

A partir desta simulação inicial os passos seguintes passaram por adicionar mais elementos à simulação para aumentar a sua complexidade. Ao mesmo tempo foi também criada uma interface web que permite visualmente apresentar o resultado da simulação.

Capítulo 2

Sistema

2.1 Arquitetura do sistema

Tendo em conta toda a discussão e estudos realizados para compreender o projeto, foi obtida a arquitetura representada na Figura 2.1 que conceptualmente pode ser dividida em três partes:

- A parte principal do projeto, que é presentada pela estrutura do protocolo de rede veicular, desenvolvido com as tecnologias *Python* em conjunto com *vanetza*, que no fundo implementa toda a parte protocolar da comunicação entre veículos.
- Os *scripts* para gerar as simulações e definir o comportamento dos vários elementos da rede, também estes escritos em linguagem *Python*.
- Um servidor web que permite uma visualização gráfica da simulação, desenvolvido em *Flask* com *Leaflet* para suportar a apresentação do mapa.
- Á parte de todos estes componente foi também construída uma base de dados para suportar a apresentação das posições dos carros e lidar com ocupação dos parques.

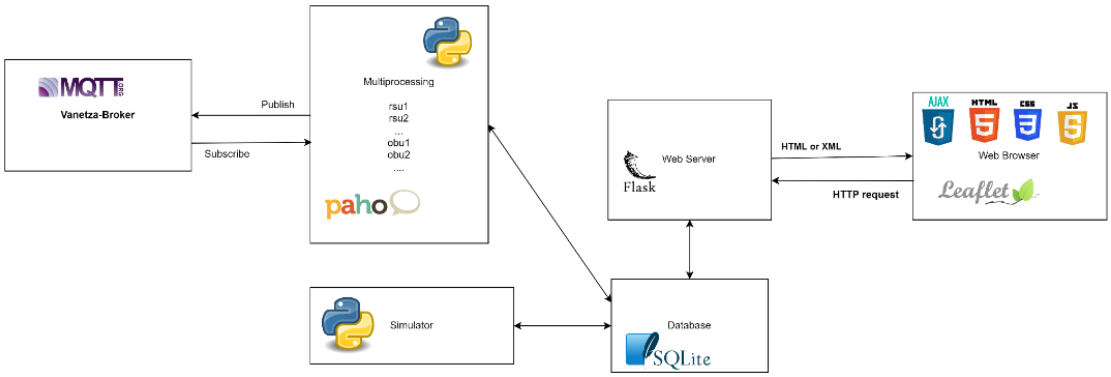

Figure 2.1: Arquitetura geral do Sistema

2.1.1 Vanetza e Brokers MQTT

O protocolo de rede veicular corre sobre o v*anetza* e no fundo consiste em lançar um conjunto de containers *Docker*, sendo que cada um vai representar uma das entidades da rede (OBU ou RSU). Para a comunicação entre containers foi usada uma rede virtual do *Docke*r (*vanetzalan0*, 192.168.98.0/24) A descrição contida no ficheiro *docker-compose.yml* permitiu especificar o número de entidades que participam na simulação (neste caso 6, 2 RSUs e 4 OBUs). Cada entidade contem o seu IP, assim como ID e endereço MAC e o tipo de entidade (15 para OBUs e 5 para RSUs). Seguidamente são apresentados todos os ips associados a cada entidade, assim como o seu respetivo ID e MAC.

- rsu1 - (192.168.98.10), 1, 6e:06:e0:03:00:01
- rsu2 - (192.168.98.20), 2, 6e:06:e0:03:00:02
- obu1 - (192.168.98.30), 3, 6e:06:e0:03:00:03
- obu2 - (192.168.98.40), 4, 6e:06:e0:03:00:04
- obu3 - (192.168.98.50), 5, 6e:06:e0:03:00:05
- obu4 - (192.168.98.60), 6, 6e:06:e0:03:00:06

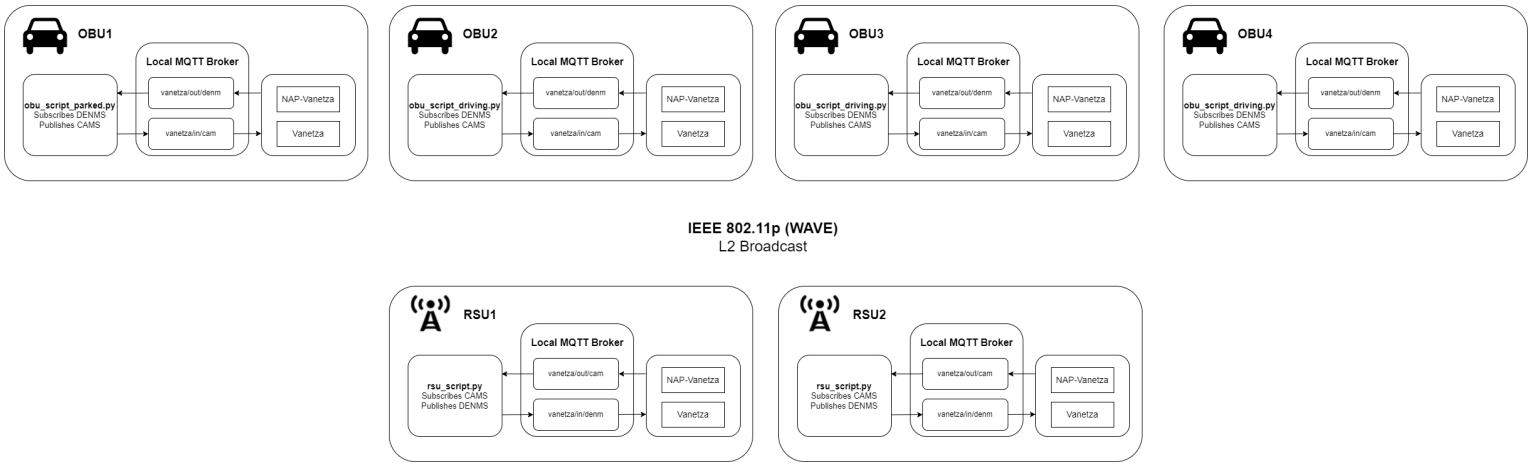

Figure 2.2: Diagrama da estrutura dos containers

Analisando a Figura 2.2 verificamos que cada container tem a si associado internamente um broker local. Ao mesmo tempo verificamos também que cada OBU efetua a subscrição de um tópico *vanetza/out/denm* para receber as *DENMs* publicadas pelas RSUs e pública mensagens no tópico *vanetza/in/cam* para enviar *CAMs* ás RSUs a notificar da sua presença.

Quase que inversamente, os containers que contém as RSUs efetuam a subscrição ao tópico *vanetza/in/denm* no qual publicam as *DENMs* que têm como destino as OBUs e subscrevem o tópico *vanetza/out/cam* para receberem as *CAMs* das OBUs

2.1.2 Scripts para geração da simulação

Tal como referido anteriormente, para a geração da simulação do comportamento dos elementos da simulação, foram desenvolvidos um conjunto de scripts de suporte em *Python*:

- ***driving.py*** contém um conjunto de funções que permitem especificar a deslocação das várias OBUs, qual o circuito que elas percorrem, em que parque estacionam, que posições percorrem quando saem do parque, etc. Por cada posição percorrida neste ficheiro, é publicada um *CAM* no tópico MQTT e efetuada uma escrita dessa respetiva posição na base de dados (para apresentação na interface web).

- ***obu_ script.py*** contém a inicialização das OBUs. Foram utilizadas threads (uma para cada obu) que são lançadas concorrentemente e efetuam múltiplas chamadas ás funções definidas no ficheiro *driving.py* para que se possa simular o comportamento da OBU. Neste ficheiro são definidos os percursos percorridos pelas OBUs assim como o processamento das *DENMs* recebidas e a tomada de decisão em relação a estacionar ou não.

- ***rsu_ script.py*** contém a inicialização das RSUs e tal como o anterior, também utiliza threads para concorrente especificar o comportamento das várias RSUs. Contém também funções adicionais de acesso à base de dados para registo de estacionamentos e de ocupação dos parques. Contém também a construção das *DENMs* a enviar à OBUs assim como o processamento das *CAMs* recebidas.

2.1.3 Aplicação Web

O servidor web, como já foi referido no inicio do capítulo, foi desenvolvido com recurso à framework *Flask*, e apresenta conteúdo no URL *http://http://localhost:3000/*.

O estilo e estrutura da página foi construído em *HTML*, *CSS* e *Javascript*, sendo que o mapa interativo apresentado foi desenvolvido com recurso á biblioteca *Leaflet*.

O acesso á base de dados faz-se do lado do servidor, para obter os dados das posições periodicamente atualizadas durante no decorrer simultâneo da simulação, que comunica para o client com ajuda da tecnologia *Ajax*. O acesso á base de dados ocorre com frequência máxima de geração standard das mensagens CAM que equivale a uma frequência de 10 Hz.

Para apresentação dos vários elementos no mapa foram criados ícones para cada nó da rede usando as especificações do *Leaflet*. A tecnologia *Ajax* em conjunto com o javascript permite efetuar vários pedidos periódicos de *HTTP GET* pela nova posição dos nós da rede sem ser necessário efetuar o reload total do mapa.

Na Figura 2.3 encontra-se representada a arquitetura lógica da aplicação web.

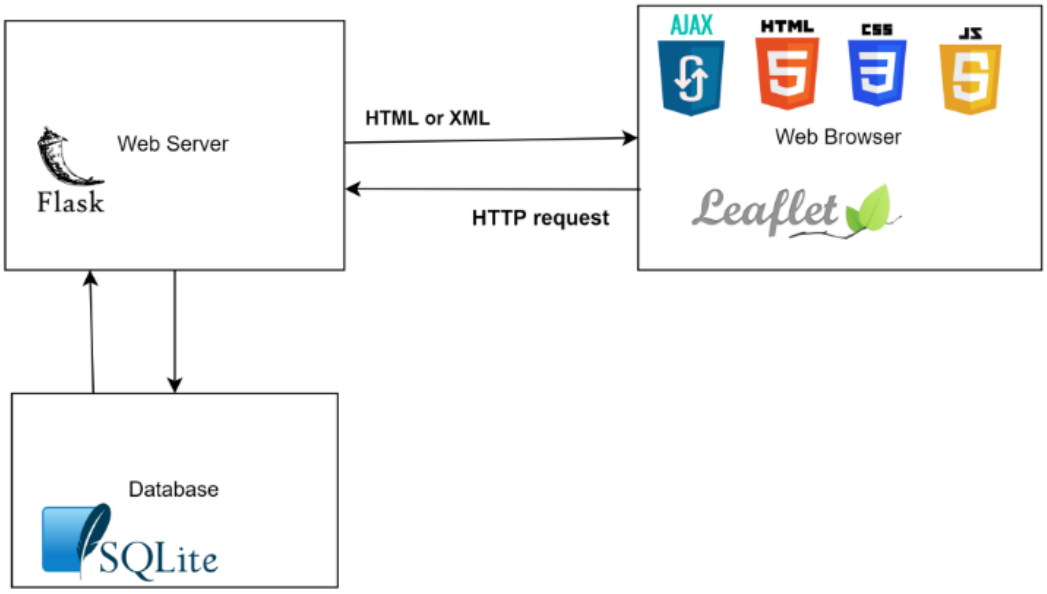

Figure 2.3: Estrutura do servidor web

2.1.4 Base de dados

A base de dados é composta por 4 tabelas: Coordenadas, Parque, RSU e OBU.

A tabela Coordenadas armazena posições geográficas dos valores latitude e longitude. Estas posições geográficas são relacionadas com os estacionamentos da tabela Parque. A tabela Parque por sua vez permite armazenar não só guardar a posição de cada lugar presente no parque de estacionamento, como ainda possui uma flag que indica se o mesmo se encontra ocupado ou não por um carro.

A tabela RSU permite armazenar informação relativa ás RSUs. É nesta tabela que se faz o mapeamento de RSU e parque, ou seja é definido sobre que parque está a RSU a monitorizar. Os dados presentes nestas três tabelas são estáticos e definidos previamente pelas entidades que ficam responsáveis pela gestão desses mesmos dados.

A tabela OBU armazenas dados usados somente para efeitos de simulação, os dados vão sendo atualizados durante o decorrer da mesma, de forma a que seja possível acompanhar o movimento dos veículos. O servidor web descrito anteriormente lê as sucessivas posições desta tabela para depois apresentar visualmente na página web.

Na Figura 2.4 está representado o diagrama da base de dados implementada.

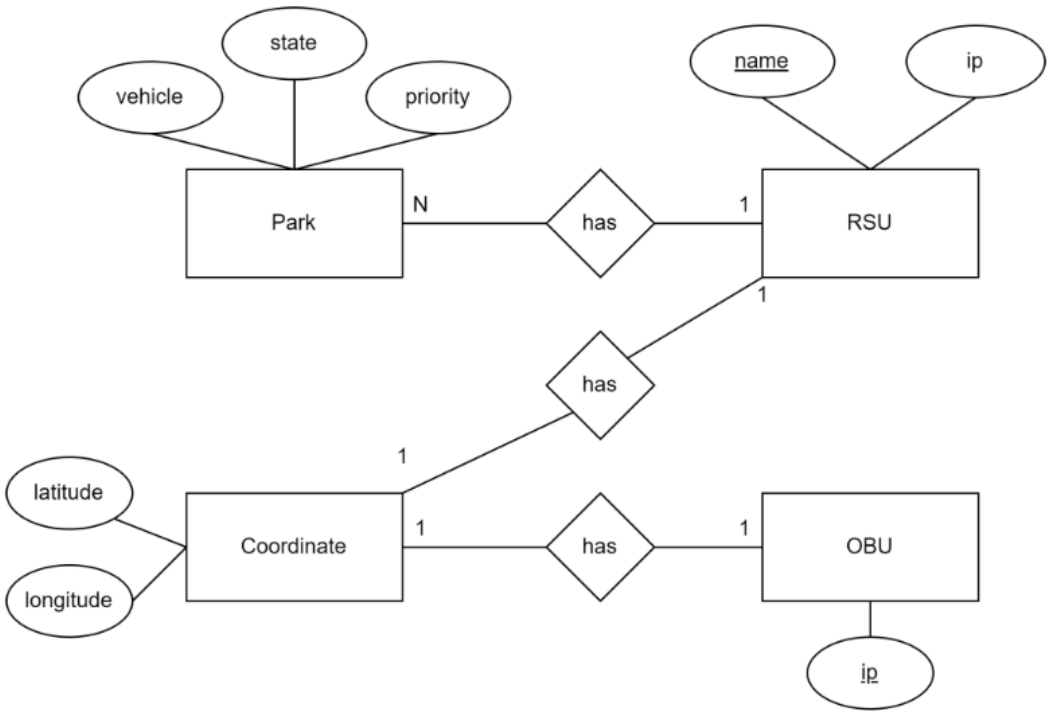

Figure 2.4: Diagrama da base de dados

Capítulo 3

Simulação

3.1 Simulação

3.1.1 Aleatoriedade

De modo a garantir que a simulação não corre sempre da mesma maneira, foram introduzidos mecanismos de aleatoriedade. Isto foi feito com recurso à definição de percursos a percorrer pelas OBUs e pela geração de números aleatórios. De acordo com o número gerado, é escolhido qual o caminho que as OBUs irão percorrer.

Foram definidos dois circuitos a serem percorridos, sendo que um deles termina com a tentativa de estacionamento no parque de estacionamento 1, enquanto que o outro termina com a tentativa no parque 2. No entanto, de modo a demonstrar todo o processo de receção de *DENMs* que especificam a ocupação do parque, caso uma das OBUs não consiga encontrar lugar num parque dirige-se ao outro para tentar estacionar. As OBUs 1, 2 e 3 percorrem um dos circuitos descritos anteriormente e terminam num lugar num dos parques de acordo com o número aleatório gerado. No entanto para aumentar ainda mais a aleatoriedade da simulação foi criada uma OBU 4 cujo ciclo de vida se inicia como estando estacionada no parque 1. Esta OBU espera um tempo aleatório e tenta ir estacionar no outro parque, pelo que se não conseguir volta ao mesmo aonde começou a simulação caso haja ainda haja um lugar livre.

Os circuitos e parques referidos anteriormente vão ser apresentados na secção Circuitos.

3.1.2 Circuitos

Na Figura 3.1 estão representados 4 pontos que são as várias etapas que uma OBU percorre ao longo do circuito 1, ou seja uma OBU que irá realizar o circuito 1, inicia o seu percurso no ponto 4, desloca-se ao ponto 1, depois ao ponto 2 e ao pronto 3 sucessivamente e quando passa na reta (ponto3 - ponto4 à entrada do parque de estacionamento verifica se pode ou não estacionar e, caso não o possa fazer segue em frente.

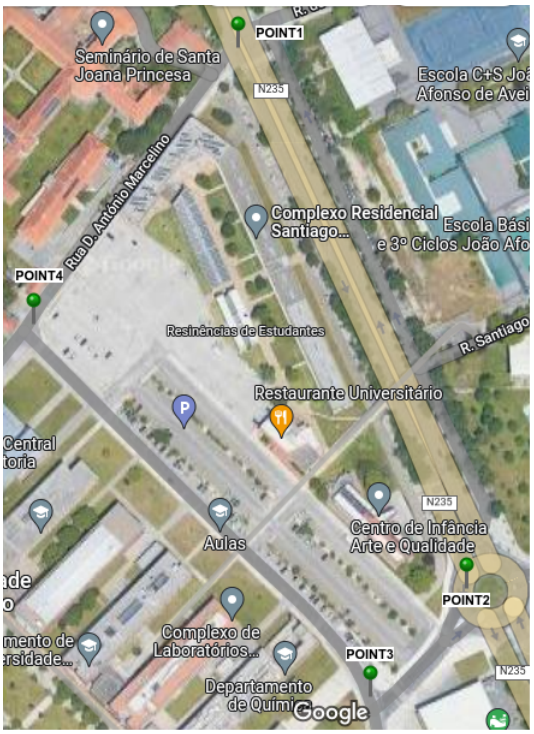

Figure 3.1: Circuito 1

Figure 3.2: Parque1

O parque 1 situado a meio da reta definida entre o ponto 3 e o ponto 4 está representado na Figura 3.2. Por uma questão de simplicidade e conveniência, foi definido que este parque teria apenas 3 lugares representados na figura. A RSU que por sua vez monitoriza a ocupação deste parque está colocada à entrada do mesmo.

O circuito 1 no fundo tem como principal objetivo que os carros efetuem uma deslocação e tentem estacionar num lugar no parque 1 (se houver disponibilidade).

Por sua vez o circuito 2 (Figura 3.3) já tem como objetivo que os carros estacionem no parque de estacionamento 2 (Figura 3.4). Caso os carros não consigam encontrar estacionamento neste parque dirigem-se ao ponto 6 e posteriormente ao ponto 1 para caso seja necessário iniciarem o circuito 1 para procurar lugar no outro parque.

A deslocação ao longo destes circuitos, assim como o processo de estacionar num parque e num lugar é feito com recurso a chamadas a funções descritas no ficheiro *driving.py*.

Figure 3.3: Circuito 2

Figure 3.4: Parque2

3.2 Interação e processamento de mensagens

RSU

A interação entre nós da rede no fundo assenta num princípio bastante simples, se uma RSU detetar uma CAM recebida dentro de uma certa área responde ao emissor com uma DENM a indicar a ocupação do parque.

Por isso sempre que é recebida uma CAM, uma RSU verifica se a velocidade presente na CAM é zero, e caso seja, e o veículo em questão não esteja registado como estacionado, é efetuado o acesso à base de dados para esse registo. No entanto é importante ter em atenção que este registo só é efetuado caso a OBU em questão esteja num lugar de estacionamento de um parque, pelo facto de poder ocorrer a situação em que uma OBU está parada na estrada. Caso a velocidade recebida seja diferente de zero então a RSU responde com uma DENM a indicar ocupação do parque. Caso a velocidade seja diferente de zero e a posição seja um dos lugares do parque então significa que a OBU vai sair do lugar, ou seja é necessário realizar o acesso à base de dados para registar esta saída e manter a consistência do número de ocupação do parque.

Finalmente, falta apenas referir que se uma CAM recebida não se enquadrar na área definida para a RSU, esta é ignorada e não é produzida nenhuma DENM de resposta. Este

mecanismo foi desenhado de forma a simular o que na prática seria alcance das RSUs.

OBU

No que diz respeito à OBU, já não existe o assíncronismo no envio de mensagens, pelo que estas são periódicas e têm uma frequência de 10Hz. Quando uma OBU recebe uma DENM, é atualizada uma variável booleana interna à simulação que indica se a mesma pode estacionar num parque ou não (de acordo com o que foi especificado na DENM).

Quando as OBUs chegam perto da entrada do parque é verificada a variável interna referida anteriormente, e se estiver com valor *True* então significa que é possível efetuar o estacionamento nesse parque. Caso contrário a OBU pode ignorar este parque e seguir para a procura de outro. Se houver lugares disponíveis para a OBU então internamente é feito um acesso à base de dados e atribuído um lugar para estacionamento deste veículo. Este procedimento ocorre apenas para que se possa gerir melhor os parâmetros da simulação, porque em situações reais ainda é necessário efetuar a procura por um lugar enquanto que na simulação a partir do momento em que uma entra dentro de um parque já tem lugar atribuído.

3.3 Ficheiros e Funções da Simulação

Ficheiro	**Descrição**
obu_script.py	Lançamento das threads que vão representar e efetuar o movimento das OBUs
rsu_script.py	Lançamento das threads que vão representar o comportamento das RSUs
driving.py	Contém o código da permite especificar o movimento das obus ao longo das várias posições dos circuitos.
driving.json	Ficheiro JSON que tem uma CAM que é usada como template pelos outros ficheiros
denm.json	Ficheiro JSON que tem uma DENM que é usada como template pelos outros ficheiros

Tabela 3.1: Ficheiros de simulação, na pasta obus_rsus

Ficheiro	Descrição
on_connect e on_disconnect	Funções para ligação ao broker e término da mesma.
on_message	Função que especifica o que deve acontecer sempre que é recebida uma mensagem pela RSU
verflocal	Verificação se um certo ponto se encontra dentro de uma certa área.
verfFreePark	Verificar se existe um lugar livre no parque sobre controlo da RSU. Veriricação é feita com um acesso à base de dados.
parkin	Registar um estacionamento na base de dados
parkout	Registar um carro a sair de um estacionamento na base de dados.
sendDenm	Construção e envio de uma DENM para o broker.
rsu_process	Especificação do comportamento da RSU.
rsu_init_simul	Lançamento das threads e inicialização da simulação.

Tabela 3.2: Funçõe do ficheiro rsu_script.py

Ficheiro	Descrição
on_connect e on_disconnect	Funções para ligação ao broker e término da mesma
on_message	Função que especifica o que deve acontecer sempre que é recebida uma mensagem do broker
get_spot_free_spotnum	Acesso à base de dados para tentar obter as coordenadas de um lugar livre num parque associado a uma RSU
course1 e course2 e course 3	Implementação dos circuitos 1 e 2. Circuito 3 consiste numa OBU que está inicialmente estacionada no parque 1 e irá procurar lugar noutro parque
obu_process	Especificação do comportamento das threads (OBUs)
obu_init_simul	Lançamento das threads e inicialização da simulação.

Tabela 3.3: Funçõe do ficheiro obu_script.py

Ficheiro	Descrição
update_db	Atualização da posição de uma OBU. Escrita dessa posição na base de dados para ser lida e apresentada no frontend
drive_in_square	Percorrer os vários pontos do circuito 1
go_to_park	Dirigir-se à entrada do parque de estacionamento 1, efectuando o percurso ponto 4 -> ponto 1 -> ponto 2 -> ponto3 -> parque.
go_straight	Estando na entrada do parque 1, dirigir-se ao ponto 4.
leave_park	Sair do parque de estacionamento 1 e ir para o ponto 4.
go_to_park2	Estando no ponto 4 dirigir-se à entrada do parque de estacionamento 2 passando pelo ponto 5.
park2	Estacionar no parque de estacionamento 2.
go_back_tostart	Estando na entrada do parque de estacionamento 2, dirigir-se ao ponto 4, passando antes pelo ponto 6.

Tabela 3.4: Funções do ficheiro driving.py

3.4 Condições de simulação

Condições Iniciais

As possíveis condições iniciais de simulação são no fundo a atribuição dos vários percursos a cada OBU, por exemplo pode acontecer que seja atribuído a todas as OBUs o mesmo percurso, ou um percurso diferente para cada uma. No caso das OBUs 1 a 3, estas percorrem sempre ou o circuito 1 ou 2.

Em relação à OBU4 esta realiza sempre o mesmo percurso que no fundo é após esperar um tempo aleatório decide sair do seu lugar (que é no parque 1) e ir procurar ao parque 2 por um lugar.

Condições esperadas

É esperado que o resultado das simulações no fim seja todos os carros terminarem estacionados num lugar de um dos parques. Em relação à OBU4 pode terminar tanto no mesmo parque em que estava inicialmente como no parque 2 pelo facto de ter conseguido arranjar lugar livre.

As OBUs 1 a 3 tanto podem terminar num lugar do parque 1 como num lugar do parque 2, mas é expectável que acabem estacionadas.

Capítulo 4

Resultados

Em termos de resultados obtidos, foi possível realizar múltiplas simulações e verificar que múltiplas "runs"seguidas oferecem diferentes resultados. Os carros percorrem percursos diferentes se correrrmos as simulações várias vezes e tentam estacionar em parques ou lugares diferentes. Verificou-se que as RSUs efectuam o processamento correto das CAMs recebidas pelos carros e que enviam a informação correta, relativa à ocupação dos parques.

Por sua vez foi também verificado que os carros se comportam de maneira correta de acordo com a receção da informação das DENMs isto é, se for indicado que não à lugares livres num parque eles seguem em frente para procurar noutro lugar, caso contrário tentam estacionar no parque.

De acordo com o que foi descrito na secção Seção 3.4, após a realização de várias simulações e análise do comportamento das obus e resultados finais, observou-se que se verificaram as condições esperadas de término de simulação.

Capítulo 5

Conclusão

5.1 Conclusões

Para o projeto de rede veicular implementado podemos retirar algumas constatações relativo ás necessidades da rede, as características dos intervenientes da rede, das OBUs e principalmente RSUs, e tratamento das mensagensn CAMs e DENMs.

A necessidade constante da rede reagir ás atualizações dos valores nos parâmetros do high frequency container, e rapidamente se adequar ao novo estado do meio, de forma, a garantir o bom funcionamento do serviço permite-nos fazer uma reflexão sobre a velocidade e resiliência necessária para estas comunicações, pelo facto de não ser aceitável as mensagens chegarem com grandes atrasos em ambientes reais. Importância essa que aumenta quando é possível evitar acidentes e salvar vidas em ambientes rodoviários.

Podemos perceber a dificuldade de gestão o evento da atualização do número de vagas de estacionamento através das mensagens de DEMN emitidas por cada RSU. Para a RSU, como responsável pela gestão do serviço, é importante que possua uma boa capacidade de acesso á rede, com pouca latência e um alcance de comunicação capaz de englobar todo o parque de estacionamento e ruas envolventes, e com um processamento projetado de acordo com a dimensão do tráfego que passa na sua região.

É importante também referir que os parques sobre os quais as RSUs têm controlo devem ser administrativamente configurados nas mesmas, pois era bastante difícil considerar que a partir de mensagens de carros parados as RSUs conseguiriam identificar e construir, só por si, o mapa de todo o parque. Assim era permitido um maior controlo sobre a área de "vigilância"da RSU assim como uma redução na complexidade do processamento que tem de ser efetuado pela mesma.

5.2 Trabalho Futuro

Os passos seguintes do projeto passaria por tornar o processo de estacionamento mais inteligente, tratando de diferentes tipos de veículos, para além do veiculo ligeiro, como por

exemplo verificar lugares de estacionamento de motas e de veículos pesados.

Seguindo ainda com a ideia de mais inteligência na gestão dos lugares, o protocolo poderia também tratar das prioridades dos estacionamentos, isto é, conferir se os veículos e os lugares livres correspondem a lugares de prioridade para grávidas e/ou deficientes.

Apesar de não ter sido implementado em ambiente de simulação, foram tomados cuidados especiais para as ideias referidas nos parágrafos anteriores. A base de dados criada contém campos para registar veículos de tipos especiais e as funções de registo de estacionamento contém parâmetros que permitem diferenciar o tipo de veículo.

Outro avanço futuro, não permitido pelas limitações do vanetza, seria acrescentar unidades VRU e mensagens VAM, de forma a conseguir gerir estacionamentos de bicicletas ou outro tipo de veículos do género.

Finalmente foi desenvolvido também um script adicional que fornecidos números de OBUs e RSUs gera um *docker-compose.yml* passível de ser usado com o ambiente do *vanetza*. O objetivo deste ficheiro era criar uma simulação totalmente parametrizável por parte dos utilizadores em que se pudessem dinamicamente criar várias unidades e observar o comportamento das mesmas. Esta ideia foi, no entanto, abandonada a meio do desenvolvimento do projeto pela complexidade necessária para a implementar. Contudo, seria algo bastante interessante para se estudar na continuação do desenvolvimento deste projeto.

5.3 Vídeo

Segue em anexo com este relatório e apresentação um vídeo que mostra o resultado de uma simulação efetuadas. Uma nota importante que após o término do vídeo foram corridas múltiplas vezes o programa para executar a simulação das OBUs mas esse comando foi parado antes que se pudesse visualizar a simulação, isto porque esta ação serviu apenas para demonstrar que correndo várias vezes seguidas o programa dá origem a que as OBUs tenham de percorrer caminhos diferentes (prova da aleatoriedade do programa),