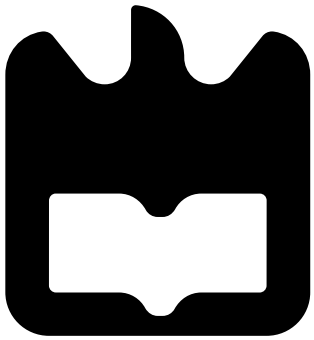
Universidade de Aveiro Departamento de Electrónica, Telecomunicações e Informática,
2018

Joana Andreia Paixão Conde

Estratégias para diminuição do tráfego de dados e controlo em mecanismos de Disseminação de Conteúdos em redes Veiculares



Joana Andreia Paixão Conde

Estratégias para diminuição do tráfego de dados e controlo em mecanismos de Disseminação de Conteúdos em redes Veiculares

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Engenharia de Computadores e Telemática, realizada sob a orientação científica da Professora Doutora Susana Sargento, Professora Associada com Agregação do Departamento de Eletrónica Telecomunicações e Informática da Universidade de Aveiro e co-orientação científica do Doutor Carlos Senna, Investigador do Instituto de Telecomunicações de Aveiro.

o júri / the jury

presidente / president

Professor Doutor António Luís Jesus Teixeira
Professor Associado com Agregação do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro (por delegação do Reitor da Universidade de Aveiro)

vogais / examiners committee

Professora Doutora Susana Isabel Barreto de Miranda Sargento
Professora Associada com Agregação do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro (orientadora)

Professor Doutor Rui Pedro de Magalhães Claro Prior
Professor Auxiliar do Departamento de Ciências de Computadores da Faculdade de Ciências da Universidade do Porto

agradecimentos

Gostaria de aproveitar esta oportunidade para agradecer aos meus pais e irmão, pelo seu apoio implacável ao longo do meu grau académico. Gostaria também, de mostrar meu agradecimento à professora Susana Sargento por me cativar na área de Telecomunicações durante meu quarto ano de curso, bem como pela oportunidade de desenvolver uma Dissertação sob sua orientação e permitir minha integração no grupo do NAP. Gostaria também de agradecer ao Carlos Senna, por todo o seu esforço na orientação desta Dissertação. Por último, queria de agradecer a todos os amigos(as) e colegas que me acompanharam ao longo do curso, bem como aqueles que conheci no Instituto de Telecomunicações. Gostaria de agradecer especialmente ao meu namorado Luís Tiago, pela sua constante paciência, disponibilidade e ajuda no desenvolvimento desta Dissertação.

Resumo

Desde sempre que a conetividade assume um papel importante em manter as pessoas ligadas entre si e ao resto do mundo. Tal facto não mudou nos tempos atuais, especialmente na era das novas tecnologias, onde se tem assistido à introdução de novas redes de telecomunicações. Entre elas destacam-se as redes veiculares constituídas por todo o tipo de veículos com capacidades de intercomunicação. Estas redes veiculares fornecem diversos serviços, nos quais se destacam o acesso à Internet, coleta de dados sensoriais e a distribuição de conteúdos não urgentes.

No entanto, as redes veiculares têm caraterísticas especificas, face a outro tipo de redes, tais como, o número elevado de veículos, rotas imprevisíveis e a constante perda de conectividade entre os mesmos, relevando vários desafios a solucionar. A solução encontrada para a conectividade intermitente prende-se com o uso de Delay-Tolerant Network (DTN)s, cuja arquitetura assegura a entrega de informação mesmo quando não existe o conhecimento total do percurso que esta deve percorrer, desde a origem até ao destino.

O trabalho realizado nesta dissertação foca-se na diminuição do tráfego de dados e controlo em mecanismos de disseminação de conteúdos não urgentes em redes veiculares. Atualmente, existem três estratégias já implementadas pelo grupo Network Architecture and Protocols (NAP) para distribuição de conteúdos não urgentes. Destas três, apenas uma delas se destaca, a estratégia LRBF, pela elevada taxa de entrega. No entanto, a elevada taxa de entrega força um *overhead* considerável na rede. Para combater este efeito, foram desenvolvidas quatro abordagens para diminuição do tráfego de controlo. Posteriormente a melhor abordagem (*BitArray*) foi escolhida para integrar numa nova estratégia, LRBFAdvanced. Para além disso, esta nova estratégia consistiu, também, num conjunto de otimizações efetuadas ao nível dos pacotes de dados da estratégia base, LRBF, para combater o número de pacotes de dados redundantes na rede. Tanto a nova estratégia como as abordagens desenvolvidas de diminuição de tráfego de controlo foram implementadas no software de DTNs mOVERS Emulator, e sempre avaliadas em relação à estratégia base. Posteriormente, a estratégia LRBFAdvanced, assim como a estratégia LRBF (para efeitos de comparação), foram introduzidas e testadas numa plataforma real (em ambiente controlado - laboratorial, e em ambiente não controlado - moliceiros na ria de Aveiro) para demonstrar que, de facto, o objetivo principal da LRBFAdvanced é cumprido e atingido na prática.

Após análise dos resultados obtidos conclui-se que a nova estratégia resultante proposta, denominada LRBFAdvanced, cumpriu o principal objetivo proposto, nomeadamente, a redução do *overhead* na rede veicular, tanto de controlo como de dados.

Abstract

Connectivity is playing an important role in keeping people connected to each other and to the rest of the world. This has not changed in the actual time, especially in the field of new technologies, where new telecommunication networks have been introduced. These include vehicular networks made up of all types of vehicles with intercommunication capabilities. These vehicular networks provide a variety of services, that can be Internet access, sensor data collection and the distribution of non-urgent content.

However, vehicular networks have specific characteristics when compared with other types of networks, such as the high number of vehicles, unpredictable routes and the constant loss of connectivity between them, highlighting several challenges to be solved. The solution found for intermittent connectivity concerns the use of DTNs, whose architecture ensures the delivery of information even when there is no complete knowledge of the route that it must travel, from the origin to the destination.

The work carried out in this dissertation focuses on the reduction of data and control traffic in non-urgent content dissemination mechanisms in vehicular networks. Currently, there are three strategies already implemented by the NAP group for non-urgent content distribution. Of these three, only one of them stands out, the Local Rarest Bundle First (LRBF) strategy, achieving high rate of delivery. However, the high delivery rate forces a considerable overhead on the network. To overcome this drawback, four approaches have been developed to reduce control traffic. Subsequently the best approach (BitArray) was chosen to integrate into a new strategy, LRBFAdvanced. In addition, this new strategy also consisted of a set of optimizations made at the level of the base strategy data packets, LRBF, to overcome the excessive number of redundant data packets in the network. Both the new strategy and the developed control traffic reduction approaches were implemented in the mOVERS Emulator DTNs software, and always evaluated against the base strategy. Later, the LRBFAdvanced strategy, as well as the LRBF strategy (for comparison purposes), were introduced and tested on a real platform (laboratory environment and an uncontrolled environment - *Moliceiros da Ria de Aveiro*) to demonstrate that, in fact, the objective of LRBFAdvanced is fulfilled and achieved in practice.

After analyzing the obtained results, it is concluded that the proposed new strategy, called LRBFAdvanced, fulfilled the main objective proposed, namely, the reduction of control and data overhead in the vehicular network.

Conteúdo

Lista de Figuras

Lista de Tabelas

Acrónimos

AID	Adaptive approach for Information Dissemination
API	Aplication Programming Interface
AU	Aplication Unit
BSS	Basic Service Set
CCH	Control Channel
CDN	Content Delivery Network
CPU	Central Processing Unit
CLA	Convergence Layer Adapter
DECA	Density-Aware Reliable Broadcasting Protocol
DTN	Delay-Tolerant Network
DSRC	Dedicated Short Range Communications
DV-CAST	Distributed Vehicular Broadcast
EBCD	Efficient Broadcast Using Network Coding and Directional Antennas
EDB	Efficient Directional Broadcast
EDI	Endpoint Identifier
GBF	Generalized Bloom Filter
GPS	Global Positioning System
GPSR	Greedy Perimeter Stateless Routing
FCC	Federal Communication Commission / Comissão de Comunicação Federal
HBFR	Hierarchical Bloom-Filter based Routing
HTTP	Hypertext Transfer Protocol
HUNET	Human Network
ISCC	IEEE International Symposium on Computer and Communications

IEEE Institute of Electrical and Electronics Engineers

IF Irresponsable Forwarding

IP Internet Protocol

IT Instituto de Telecomunicações

ITS Intelligent Transport Systems

IPC Inter-Process Communcation

IVC Inter-vehicle Communications

JSON JavaScript Oject Notation

LDMB Link-based Distributed Multi-hop Broadcast

LNHF Least Number of Hops First

LRBF Local Rarest Bundle First

LRBFAdvanced Local Rarest Bundle First Advanced

LRGF Local Rarest Generation First

MAC Medium Access Control

MANET Mobile Ah-Hoc NETworks

MATLAB MATrix LABoratory

MDDV Mobility-Centric Data Dissemination

MHVB Multi-hop Veicular Broadcast

MIPS Millions of Instructions Per Second

mOVERS mobile Opportunistic Vehicular Emulator for Real Scenarios

MPR Multi-Point Relay

NAP Network Architecture and Protocols

NTP Network Time Protocol

OAPB Protocolo Optimized Adaptative Probabilistic Broadcast

OBU On-Board Unit

ODMRP On-Demand Multicast Routing Protocol

OSI Open Systems Interconnection / Interconexão de Sistemas Abertos

OSNR Obfuscated Social Network Routing

PHP Hypertext Preprocessor

P2P	Peer-to-Peer
REDEC	REceiver-based solution with video transmission DECoupled
RSSI	Received Strength Indicator
RSU	Road Side Unit
SB	Smart Broadcast
SCF	Store-Carry-and-Forward
SCH	Service Channel
SDK	Software Development Kit
SPAWN	Swarming Protocol For Vehicular Ad-Hoc Wireless Networks
TCP	Transmission Control Protocol / Protocolo de Controlo de Transmissão
UDP	User Datagram Protocol
UMB	Urban Multi-hop Broadcast
UMWE	Universal WAVE Management Entity
V2I	Vehicle-to-Infrastructure
V2V	Vehicle-to-Vehicle
VANET	Vehicular Ah-Hoc NETwork
VDTN	Vehicular Delay-Tolerant Network / Redes Tolerantes a Atrasos Veículares
WAVE	Wireless Access for Vehicular Environments
Wi-Fi	Wireless Fidelity
WLP	Wu and Li's Algorithm based Reliable Broadcasting Protocol
ZMQ	Zero-M Queue

Capítulo 1

Introdução

O presente documento foi desenvolvido no âmbito da Dissertação de Mestrado em Engenharia de Computadores e Telemática do Departamento Eletrónica, Telecomunicações e Informática da Universidade de Aveiro, com o tema "Estratégias para diminuição do tráfego de dados e controlo em mecanismos de Disseminação de Conteúdos em redes Veiculares".

Neste capítulo são apresentados o contexto e motivação desta Dissertação bem como os principais objetivos e contribuições. Por último, é feita uma breve descrição sobre a organização do presente documento.

1.1 Contexto e Motivação

Uma cooperação entre as Universidade de Aveiro, e do Porto, o Instituto de Telecomunicações (IT), juntamente com a Veniam [15], levou à implementação de uma ampla rede veicular, a nível mundial, com mais de 600 veículos (autocarros, táxis, camiões do lixo), e infraestruturas fixas, localizada na cidade do Porto. Esta plataforma de comunicação é completamente compatível com as normas que regulam as comunicações veiculares (IEEE 802.11p/IEEE 1609 - WAVE).

As redes veiculares têm caraterísticas especiais como a mudança bastante rápida da topologia da rede, ligações intermitentes e frequentes disrupções da rede, devido à mobilidade elevada dos nós da rede [16]. De forma a lidar com estes desafios é utilizado o conceito de DTN [17] que introduz o mecanismo de *store-carry-forward* de forma a assegurar a entrega de informação, quando não está definido nenhum caminho *End-to-End*. Assim, através deste mecanismo, a transmissão de informação em ambientes esparsos é feita por nós da rede (veículos) atuando como *data mules*. Como resultado, as DTNs tornam-se numa boa alternativa para lidar com os requisitos das redes veiculares, sendo designadas por Vehicular Delay-Tolerant Network / Redes Tolerantes a Atrasos Veículares (VDTN), quando aplicadas às mesmas.

A Figura 1.1 ilustra a arquitetura das redes veiculares. Todos os veículos dentro do alcance da infraestrutura fixa recebem informações. Posteriormente, estas informações são compartilhadas entre todos os veículos vizinhos dentro do alcance do veículo transmissor. Os conteúdos enviados consistem em atualizações de *firmware* dos equipamentos, informações de interesse turístico sobre locais onde os veículos se encontram, e também pode ser utilizada para disseminação de publicidade, por exemplo.

As estratégias de disseminação de conteúdos não urgentes atingem uma taxa de entrega elevada, estando diretamente relacionada com o processo de selecionar a informação correta

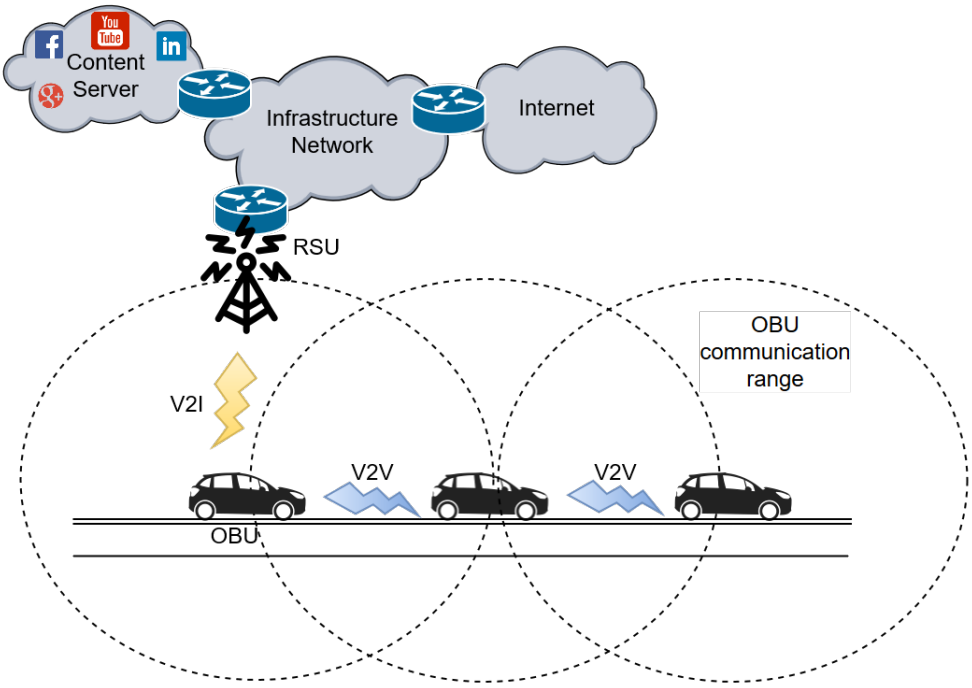


Figura 1.1: Arquitetura de uma Rede Veicular.

para ser enviada. Para desempenhar este processo de seleção, um nó tem de saber quais os pacotes que deve enviar, tendo em conta não só quais os pacotes que são importantes para um só nó especifico, mas sim para a maioria dos seus nós vizinhos. Como tal, é introduzido um novo tipo de pacote, designado de *pacote de controlo*, cujo objetivo é informar os nós vizinhos sobre o conteúdo de informação que um determinado nó tem armazenado, para que os pacotes de dados sejam distribuídos com a máxima eficiência. Para isso, é necessário incluir no pacote uma lista completa dos pacotes que o nó já tem até ao momento, podendo levar a um tamanho excessivamente grande do pacote de controlo, dependendo da quantidade de informação que já tem armazenada.

Como resultado, a introdução destes pacotes de controlo introduz *overhead* na rede, podendo esta ficar "entupida"com informações não relacionadas com as que os nós de destino desejam.

Também, nestas estratégias são usados protocolos *broadcast multi-hop*, onde a informação é espalhada na rede através da técnica de *flooding* ("inundação"da rede), isto é, todos os veículos enviam pacotes para todos os seus vizinhos, tendo como objetivo alcançar todos os veículos pertencentes à rede, levando ao aparecimento do problema de *Broadcast Storm* [18]. Como resultado, a quantidade de pacotes de dados redundantes que circulam na rede aumenta significativamente, introduzindo *overhead* na rede.

O objetivo principal desta dissertação prende-se com o desenvolvimento de abordagens para diminuição do tráfego de dados e de controlo em mecanismos de disseminação de conteúdos em redes veiculares, diminuindo assim, o *overhead* introduzido na rede, quer pelo tamanho excessivo dos pacotes de controlo quer pelo número de pacotes de dados redundantes.

1.2 Objetivos e Contribuições

O principal objetivo desta Dissertação é o estudo e implementação de abordagens de otimização da disseminação de conteúdos através de uma rede veicular, com o mínimo de sobrecarga de rede. Como tal, os objetivos são os seguintes:

- Análise estatística da rede veicular real uma vez que, são usados dados da rede veicular real implementada no Porto, podendo ser analisada para melhor entendimento da mobilidade dos veículos e comportamento da rede;
- Estudo de estratégias de disseminação de conteúdo previamente implementadas;
- Conceção e implementação de abordagens focadas na redução de sobrecarga de dados e de controlo em mecanismos de disseminação de conteúdos em redes veiculares;
- Testes e análise das abordagens implementadas;
- Teste da melhor abordagem numa rede veicular real, em laboratório (ambiente controlado) e nos moliceiros da ria de Aveiro (ambiente não controlado).

As contribuições desta dissertação são as seguintes:

- Testes de emulação e reais e avaliação do desempenho: apresentar uma demonstração clara do conceito e viabilidade da abordagem de otimização da distribuição de conteúdos mais adequada.
- Várias abordagens de otimização da distribuição de conteúdos com diminuição do *overhead* na rede reduzida foram propostas, implementadas e avaliadas.
- Foi feita a adaptação da largura de banda aos pacotes de controlo, o que tornou possível a execução do cenário de *parking*.
- Foi proposta uma nova estratégia com a melhor abordagem de otimização de controlo e redução da quantidade de dados redundantes na rede.
- Como resultado do trabalho desta dissertação foi escrito e apresentado um artigo com o título *Content Distribution Optimization Algorithms in Vehicular Networks*, na conferência IEEE International Symposium on Computer and Communications (ISCC), Natal, Brasil, Junho 2018.
- Encontra-se em preparação um artigo de revista com o conjunto das estratégias de disseminação que permitem diminuir o número de pacotes de dados e o tamanho dos pacotes de controlo da rede.

1.3 Estrutura do Documento

Esta dissertação está organizada em sete capítulos:

- **Capítulo 1** - *Introdução* - Este capítulo descreve o contexto e a motivação desta dissertação, bem como os objetivos, contribuições e a sua estrutura.

- **Capítulo 2** - *Conceitos Fundamentais* - Este capítulo apresenta uma análise geral e a descrição dos conceitos fundamentais relacionados com esta dissertação: redes veiculares e redes tolerantes a atrasos.

- **Capítulo 3** - *Trabalhos Relacionados* - Este capitulo apresenta uma pesquisa na literatura sobre abordagens de otimização do tráfego de controlo e dados em mecanismos de distribuição de conteúdos em redes veiculares, onde a mobilidade dos nós é um grande desafio.

- **Capítulo 4** - *Algoritmos de Otimização de Conteúdo nas Redes Veiculares* - Este capítulo apresenta quatro abordagens de otimização do tráfego de controlo, bem como as diversas modificações ao nível do controlo do tráfego de dados, efetuadas na estratégia base, dando origem a uma nova estratégia.

- **Capítulo 5** - *Implementação e Integração* - Este capítulo apresenta a implementação e integração das quatro abordagens de otimização de controlo de conteúdos nas redes veiculares, aplicados à estratégia mais eficiente, bem como a estratégia resultante das diversas modificações efetuadas ao nível do tráfego de dados, na plataforma mOVERS. Para isso, apresenta inicialmente uma explicação completa do funcionamento do mOVERS, e as suas principais modificações que foram necessárias efetuar.

- **Capítulo 6** - *Testes e Resultados* - Este capítulo descreve os cenários avaliados, os equipamentos e as plataformas utilizados para os testes realizados. Apresenta e discute os resultados obtidos através da experiência com o mOVERS e através de testes reais, em laboratório (ambiente controlado) e nos moliceiros da ria de Aveiro (ambiente não controlado).

- **Capítulo 7** - *Conclusões e Trabalho Futuro* - Este capítulo contém as conclusões relacionadas com o trabalho desenvolvido, e também, aponta para possíveis melhorias e orientações futuras que poderão ser desenvolvidas para melhorar e complementar esta dissertação.

Capítulo 2

Conceitos Fundamentais

2.1 Descrição do Capítulo

O foco desta dissertação abrange duas áreas fundamentais: Redes Ad-hoc Veiculares (VANETs) e Redes Tolerantes a Atrasos (DTNs). Este capítulo visa familiarizar o leitor com estes dois tópicos, fornecendo uma visão geral das definições e conceitos fundamentais associados a eles. Como tal, este capítulo está organizado da seguinte forma:

- *secção 2.2*: *Vehicular Ad-Hoc Networks* - apresenta a definição e os conceitos básicos das redes VANETs, bem como a sua arquitetura, características, desafios, aplicações, tecnologias e padrões, e também protocolos de disseminação nestas redes.
- *secção 2.3*: *Veicular Delay Tolerant Networks* - introduz o conceito de VDTN com foco nas suas características e arquitetura.
- *secção 2.4* : *Considerações Finais* - apresenta o resumo deste capítulo.

2.2 Vehicular Ad-Hoc Networks

Mobile Ad-Hoc Network (MANET) é conhecida como uma rede sem fios ad-hoc. É definida como sendo uma rede com múltiplos nós livres e autónomos que têm a capacidade de se auto-organizar de maneira a criar um rede sem fios, podendo comunicar entre si e com outros nós em qualquer lugar. *Vehicular Ad-Hoc Networks* (VANET) surgem como uma extensão de MANET para sistemas de veículos, podendo abranger aviões, comboios, barcos, automóveis e robôs [6].

2.2.1 Definição

Com o desenvolvimento rápido das comunicações sem fios e das tecnologias de transporte inteligente, do inglês, *Intelligent Transport Systems* - ITS, as redes veiculares emergiram como sendo um novo paradigma [19].

Os veículos de uma VANET são considerados nós móveis inteligentes. Estão equipados com unidades móveis que integram interfaces sem fios, designadas por *On-Board Unit* - OBUs, para permitir comunicações com outros veículos e com infraestruturas fixas ao longo das estradas. As infraestruturas são nós estáticos, posicionadas estrategicamente para melhorar

a conectividade e a entrega de serviços. Estas unidades fixas são designadas por *Road Side Unit* - RSUs

Deste modo, podem ser estabelecidos dois tipos de comunicações: comunicações de Veículo-para-Veículo, designadas por *Vehicle-to-Vehicle* -V2V ou *Inter-vehicle Communications* - IVC, quando são trocados dados entre veículos na mesma vizinhança, e comunicações de Veículo-para-Infraestrutura, designadas por *Vehicle-to-Infrastructure* -V2I, quando a troca de dados é feita entre um veículo e uma infraestrutura, estando esta última na sua área de cobertura. A Figura 2.1 mostra um exemplo de uma VANET, assim como, os dois tipos de comunicação que podem ser estabelecidos.

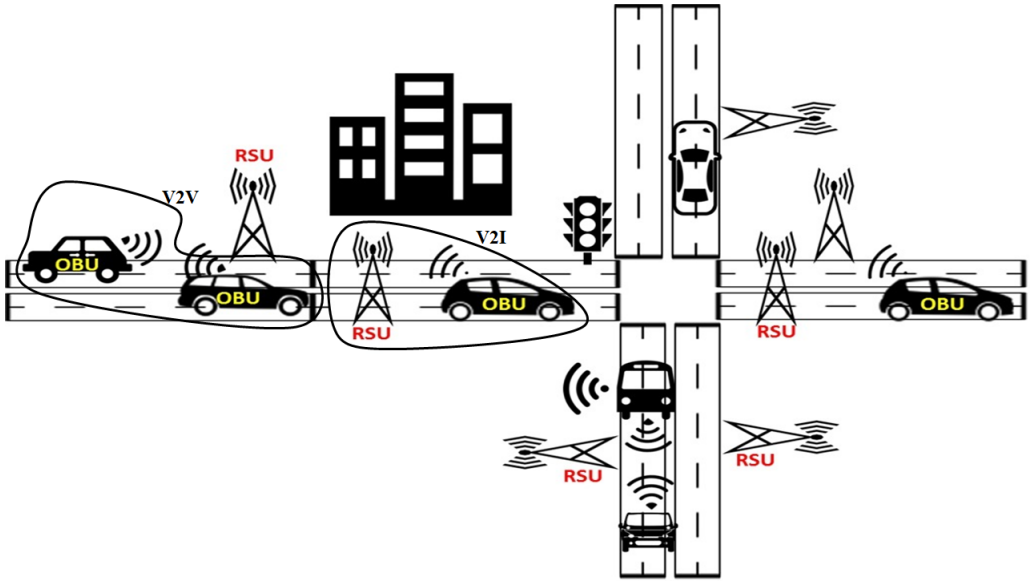


Figura 2.1: Exemplo de uma rede VANET[1].

2.2.2 Componentes da rede

As VANETs são constituídas por três componentes principais: OBUs, Unidades Aplicacionais, do inglês *Aplication Units* - (AU) e por RSUs [20].

Uma OBU é um dispositivo de rede para comunicação sem fios de curto alcance para meios veiculares, com base na tecnologia *Wireless Access for Vehicular Environments* - WAVE (IEEE 802.11p). É instalado no interior dos veículos, e é usado para trocar informações com as RSUs e/ou com outras OBUs. Está equipado com *Central Processing Unit* - CPU, memória e recursos de armazenamento e com outros interfaces de comunicação sem fios, como IEEE 802.11a/b/g ou tecnologia celular. As principais funções da uma OBU são o acesso rádio sem fios, encaminhamento ad hoc, controle do congestionamento de rede, transferência confiável de mensagens, e segurança de dados.

A AU é usada para a execução de aplicações utilizando as capacidades de comunicação de uma OBU. Pode ser ligada à OBU através de uma ligação com ou sem fios e pode residir juntamente com a OBU na mesma unidade física, sendo a diferença entre estas duas apenas lógica. A AU comunica com a rede exclusivamente através da OBU.

A RSU é uma infraestrutura fixa posicionada ao longo das estradas de forma a melhorar a conectividade e a entrega de serviços. Está equipada com uma ou mais interfaces de comu-

nicação, para comunicar com veículos, usando as tecnologias sem fios, como IEEE 802.11p ou IEEE 802.11a/b/g, e para comunicar com a rede de infraestrutura, usando a tecnologia com fios, como Ethernet ou fibra ótica. As principais funções de uma RSU são redistribuição e envio de informação de ou para outras RSUs/OBUs, fornecer o acesso à Internet ás OBUs (funcionando como *gateway*) e execução de aplicações seguras atuando como origem da informação.

2.2.3 Caraterísticas

Segundo [2], as características das VANETs são as seguintes:

- Mobilidade Elevada: É uma das características mais importantes. Os nós da rede estão em constante movimento durante a maior parte do tempo, com diferentes velocidades e direções. De acordo com [21, 22], a mobilidade elevada dos nós reduz o tempo de contacto entre eles, havendo menos rotas entre eles. Em comparação com as MANET, a mobilidade nas VANET é maior.

- Topologia dinâmica: Dada a mobilidade elevada, a topologia da VANET está em constante mudança, sendo dinâmica e imprevisível. Os tempos de ligação são curtos, especialmente entre nós que se deslocam em direções opostas. Esta topologia facilita o ataque a toda a rede e dificulta a deteção de mau funcionamento.

- Desconexões frequentes: A topologia dinâmica e a mobilidade elevada dos nós, bem como outras condições, como o clima, a densidade do tráfego, a presença de edifícios altos, causam desconexões frequentes dos veículos da rede.

- Largura de banda limitada: A banda *Dedicated Short Range Communication* (DSRC) padronizada (5.850-5.925 GHz) para redes VANET pode ser considerada como limitada. A largura da banda total é de apenas 75 MHz [3]. O débito teórico máximo é de 27 Mbps.

- Potência de transmissão limitada: A transmissão que os dados podem alcançar é limitada para as interfaces de comunicação WAVE. Esta distância é de até 1000 m. No entanto, para certos casos específicos, como emergência e segurança pública, é permitido transmitir a uma distância superior a 1000 m [23].

- Consumo de energia: Ao contrário dos outros tipos de redes móveis, as VANETs não sofrem de problemas de energia, pois os nós são veículos móveis.

- Disponibilidade do meio de transmissão: O ar é o meio de transmissão das redes VANETs. Embora a disponibilidade universal deste meio de transmissão sem fios seja uma das grandes vantagens da comunicação V2V, torna-se a origem de alguns problemas de segurança. Estes problemas estão relacionados tanto com a natureza da transmissão em ambiente sem fios, tanto com a segurança das comunicações usando um suporte aberto.

- Ocupação anónima da largura de banda: A transmissão de dados usando um meio sem fios é geralmente anónima. Pondo de lado as restrições e os regulamentos de utilização, qualquer pessoa equipada com um transmissor que opera na mesma faixa de frequências pode transmitir e ocupar a largura de banda.

2.2.4 Desafios

A grande maioria das características mencionadas anteriormente constituem, também, um desafio para as rede veiculares. Os desafios das redes VANET são os seguintes, segundo [20], [24]:

- **Escabilidade da rede:** A VANET pode estender-se ao longo do comprimento de toda a estrada, incluindo muitos veículos.
- **Topologia dinâmica/Mobilidade elevada dos nós:** os nós de uma VANET estão em constante movimento durante a maior parte do tempo, com diferentes velocidades e direções, levando a que a topologia da rede esteja em constante mudança. A elevada velocidade dos veículos, principalmente em cenários de autoestradas, faz com que o tempo de conexão entre os nós seja reduzido.
- **Conetividade intermitente dos nós:** a mobilidade elevada dos nós da rede leva a uma topologia altamente dinâmica e imprevisível, resultando em desconexões frequentes entre os nós, especialmente ao considerar cenários de autoestradas, onde a densidade de veículos é menor quando comparada com um cenário urbano.
- **Segurança e privacidade:** Manter um equilíbrio razoável entre segurança e privacidade é um dos principais desafios nas redes VANET; verificar a confiablidade de informações do remetente é importante para o destinatário. No entanto, esta verificação pode violar a privacidade do remetente.
- **Heterogeneidade de aplicações:** as VANETs devem ser capazes de fornecer uma grande variedade de aplicações de segurança rodoviária e/ou apenas informativas. As aplicações de segurança rodoviária, sendo o conteúdo de emergência, exigem um menor atraso na entrega e uma elevada prioridade. Em contrapartida, as aplicações informativas exigem um melhor rendimento e uma maior utilização dos recursos, mas podem proporcionar maiores atrasos na entrega da informação. Assim, é necessário desenvolver uma abordagem que garanta que ambos os serviços coexistam.

2.2.5 Dedicated Short Range Communications/WAVE:

Para as redes VANETs, a norma WAVE tem modificação em praticamente todas as camadas do modelo *Open System Interconnection* (OSI). Deve notar-se que, na literatura, muitas vezes DSRC (Dedicated Short Range Communications), WAVE (Wireless Access in Vehicular Environments) ou mesmo IEEE 802.11p são usados para designar todo o protocolo de pilha de padrões que lidam com as VANETs [2].

DSRC

Para maximizar a interoperabilidade, e com o objetivo de padronizar as frequências com as quais as VANETs funcionam, o governo dos EUA, representado pela *Federal Communication Commission* (FCC), criou a banda de 5850 a 5925 GHz (75 MHz de largura). Esta banda é conhecida como Dedicated Short Range Communications DSRC e é utilizada para melhorar a segurança e o fluxo de tráfego. O uso da banda DSRC não está sujeito propriamente a uma licença, mas sim a regras de uso. A banda DSRC é dividida em sete canais de 10 MHz,

respetivamente, com 178, 172, 174, 176, 180, 182, 184. O canal 178 é o canal CCH (*Control Channel*). Os outros seis são canais SCH (*Service Channels*). Os canais de serviço 172 e 184 são, respetivamente, reservados para *High Availability and Low Latency* (HALL), e para alta potência e segurança pública, como mostra a Figura 2.2. Na Europa, a banda DSRC é regulada pelo *European Telecommunications Standards Institute* [25], e apenas os canais 180 de CCH e 172, 174, 176, 178 de SCH são usados, como é mostrado na Figura 2.3 [2].

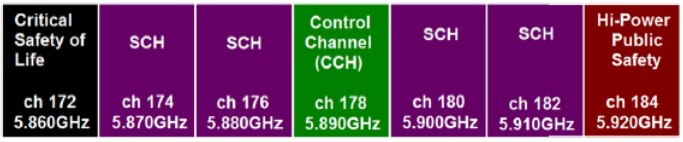


Figura 2.2: Representação da alocação dos 7 canais para DSRC nos EUA segundo [2].

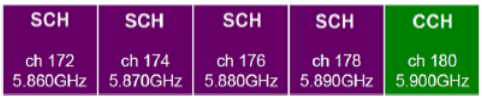


Figura 2.3: Representação da alocação dos 5 canais para DSRC na Europa, segundo [2].

Protocolo WAVE

Toda a pilha de protocolos DSRC incluindo IEEE 802.11p (camadas física e *Media Access Control* MAC) foi normalizada pelo grupo de trabalho IEEE, de forma a responder às necessidades especificas impostas pelas redes VANET, levando assim, à criação do protocolo WAVE, designado de acesso sem fios para ambientes veiculares. Assim, a pilha do protocolo WAVE é constituída por IEEE 802.11p e pela família IEEE 1609.X [3] [2], como é mostrado na Figura 2.4. Seguidamente, o objetivo e as caraterísticas especiais do IEEE 802.11p e IEEE 1609.X são explicadas.

IEEE 802.11p A norma IEEE 802.11p é uma extensão da norma IEEE 802.11, que resultou de modificações da camada física da norma IEEE 802.11a. A modificação mais importante foi a capacidade de comunicação fora da área de cobertura do *Basic Service Set*, BSS. Outra modificação relevante, mas agora ao nível da camada MAC, foi a adequação do processo de autenticação para a norma IEEE 802.11p.

Em suma, a norma IEEE 802.11p especifica as funções ao nível das camadas física e MAC necessárias para que um dispositivo IEEE 802.11 funcione num ambiente veicular.

IEEE 1603.X Define uma arquitetura e um conjunto complementar de protocolos, serviços e interfaces normalizados que permitem que todas as estações WAVE operem num ambiente veicular e estabeleçam comunicações V2V e V2I.

O WAVE não usa nenhum processo de associação, levando a que o processo de comunicação seja mais rápido, como mostra a Figura 2.5, tanto para comunicações V2V, como para comunicações V2I. Essa característica pode ser vista como uma grande vantagem e é realmente necessária num ambiente veicular, embora também possa apresentar algumas desvantagens. Devido à sua natureza de transmissão, o WAVE é vulnerável à utilização in-

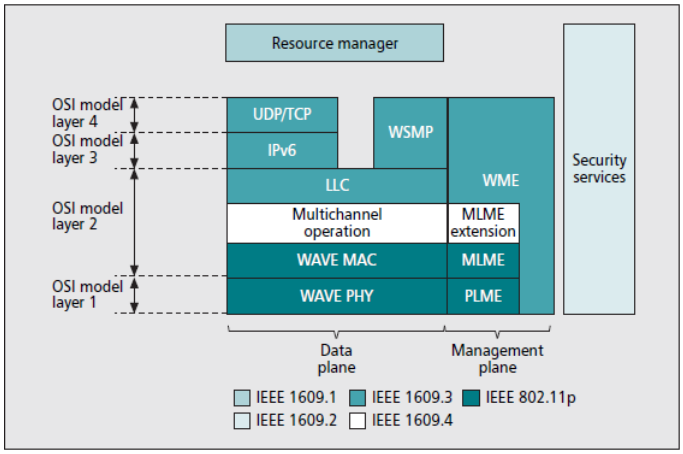


Figura 2.4: Representação da pilha do protocolo WAVE [3].

devida da largura de banda. Por exemplo, um nó pode enviar mensagens sem utilidade na transmissão e afetar negativamente todas as ligações de outros nós que estão na sua área de cobertura.

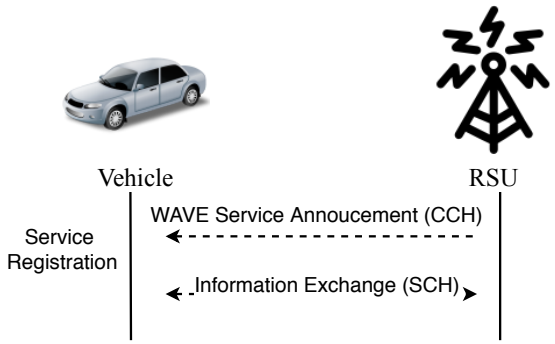


Figura 2.5: Processo de comunicação WAVE, baseado em [4].

Em suma, na Tabela 2.1 é feita um comparação entre DSRC, Wi-Fi e tecnologias celulares usados em ambientes veiculares.

	DSRC	Wi-Fi	**Sistemas Celulares**
Alcance	100 a 1000 m	30 a 100 m	1 a 30 km
Latência	200 μs	3 ms	200 ms a 3s
Custo	Sem ou barato	Sem ou barato	Caro

Tabela 2.1: Comparação entre DSRC, Wi-Fi, Tecnologias Celulares [4].

2.3 Delay-Tolerant Networks

2.3.1 Definição

As Redes Tolerantes a Atrasos, do inglês *Delay Tolerant Networks* - DTNs, são definidas como redes ocasionalmente conectadas, que podem sofrer partições frequentes. Nas redes veiculares, os veículos estão distribuídos por uma grande área geográfica, movendo-se aleatoriamente, com diferentes velocidades e trajetórias, o que faz com que a rede seja facilmente particionada. Estas características são similares às características das redes DTNs. Assim, as DTNs tornam-se numa alternativa para lidar com os principais requisitos das VANETs, sendo definidas de *Vehicular Delay Tolerant Network* (VDTN) quando aplicadas às mesmas [26].

Primeiramente, as DTNs foram concebidas para comunicações de longas distâncias, tais como comunicações inter-espaciais [27], em que o principal desafio é a latência, podendo ser horas ou mesmo dias.

A necessidade de criar tal rede, advém do facto dos protocolos da Internet serem baseados no *Transmission Control Protocol* TCP/IP [28]. O TCP/ IP não está preparado para operar em ambientes com atrasos e interrupções frequentes na rede, devido aos pressupostos fundamentais da Internet.

2.3.2 Arquitetura

As DTNs garantem a comunicação confiável entre os nós da rede, superando os problemas associados às interrupções constantes e a atrasos longos ou variáveis, através da criação de uma nova camada, entre as camadas de Transporte e a Aplicação, designada de *bundle*. Na Figura 2.6 é feita a comparação entre a pilha de protocolos Internet e a pilha de protocolos DTNs.

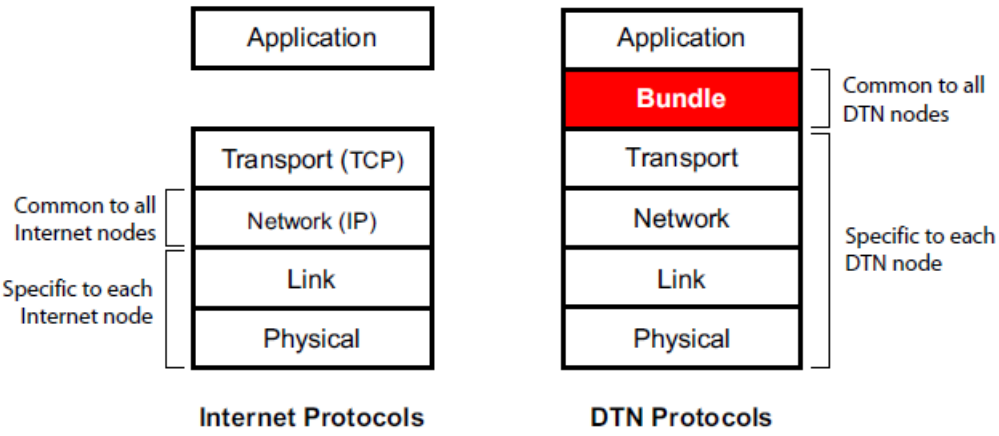


Figura 2.6: Protocolo DTN vs. Protocolo Internet [5].

De acordo com Ma *et al.* [29], a camada *bundle* permite um armazenamento persistente para lidar com interrupções de rede e inclui, também, um mecanismo de comunicação confiável que se baseia num processo de confirmação *end-to-end*. Esta camada apresenta, ainda, recursos de diagnóstico, gestão, segurança e caraterísticas de interoperabilidade.

A rede DTN introduz um novo elemento, designado de *Convergence Layer Adapter* CLA. Desta forma, as DTNs podem utilizar uma grande variedade de protocolos de entrega, no qual possuem diferentes caraterísticas e diferentes tipos de semântica nas suas mensagens. O CLA fornece as funções necessárias para transportar os *bundles* em cada protocolo subjacente levando à interoperabilidade. Estas funções estão evidenciadas na Figura 2.7.

A arquitetura mostra que existe um elemento central, designado *bundle forward*, que é responsável por fazer o encaminhamento dos *bundles* entre as aplicações, CLAs, e o armazenamento baseado em decisões de encaminhamento. Por outro lado, é responsável por efetuar a troca de informação usadas no encaminhamento, na gestão e em aplicações.

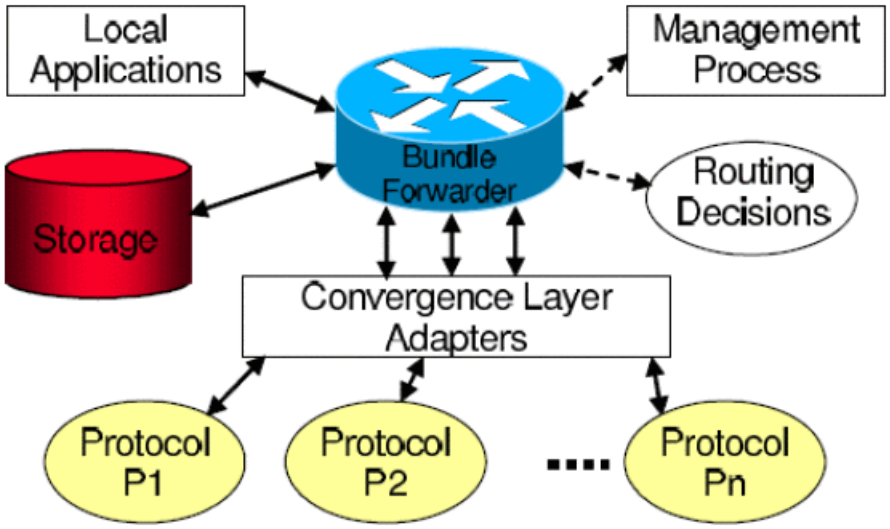


Figura 2.7: Arquitetura de encaminhamento de um *bundle* mostrando como um *bundle forward* interage com armazenamento, decisões de encaminhamento e CLA, para usar vários protocolos de entrega. [6]

2.3.2.1 Protocolo *Bundle*

Como referido anteriormente, as redes DTNs introduzem uma nova camada chamada *bundle*, de forma a conseguirem uma comunicação confiável entre os nós da rede. Esta nova camada depende de um novo protocolo normalizado por Scott *et al.* [30], designado de Protocolo *Bundle*. Todas as redes DTNs usam o mesmo protocolo *bundle*, embora possa haver variações entre os nós, dos protocolos das camadas mais baixas, dependendo do ambiente de comunicação que se enfrenta.

As principais capacidades do protocolo *Bundle* podem ser resumidas da seguinte forma, de acordo com Scott *et al.* [30]:

- Retransmissão baseada na custódia da informação;
- Capacidade para lidar com interrupções intermitentes;
- Capacidade de aproveitar ao máximo a conectividade prevista, programada e oportunista;
- Ligação tardia dos terminais da rede;

A Figura 2.8 apresenta a estrutura do protocolo *bundle*, discutido anteriormente. Por outro lado, é mostrado na Figura 2.9 que o protocolo *bundle* usa protocolos normalizados da Internet, tipicamente o protocolo TCP/IP, para efetuar o transporte e encaminhamento da informação nas camadas mais baixas. Os nós que efetuam o encaminhamento dos *bundles* podem implementar os mesmos ou diferentes protocolos nas camadas mais baixas. Estas funções podem ser comparadas a routers da Internet ou a gateways, respetivamente.

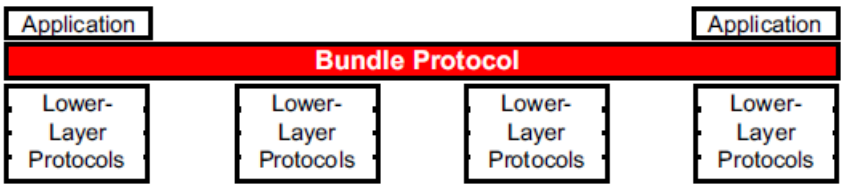


Figura 2.8: Estrutura do Protocolo-*Bundle* [5].

2.3.2.2 Mecanismo de *Store-Carry-Forward*

Para ultrapassar os vários desafios impostos pelas redes veiculares, como conectividade intermitente ou interrupções constantes, atraso longo ou variável (horas ou até mesmo dias), taxas de dados assimétricas e altas taxas de erros [5], as DTNs usam um mecanismo de SCF. Este mecanismo é apresentado na Figura 2.10. Contrariamente às redes IP que são baseadas no mecanismo de *store-forward*, as redes DTN precisam de guardar e carregar a informação enquanto não há ligação entre os nós da rede.

De forma a que os nós da rede DTN possam armazenar e carregar a informação, têm de estar equipados com um dispositivo de armazenamento, como por exemplo, um disco rígido. Esse armazenamento deve ser persistente, de acordo com F. Wartman *et al.* [5] porque:

- A ligação ao próximo salto pode não estar disponível por um longo período de tempo;
- Se ocorrer um erro aquando do envio da informação, ou simplesmente se esta for rejeitada pelo destinatário, a informação tem de ser retransmitida;
- Perante um par de nós, pode acontecer que um deles consiga enviar a informação mais rápido do que o outro.

2.4 Considerações Finais

Este capítulo apresentou uma breve introdução aos vários tópicos relativos ao principais conceitos de rede desta Dissertação: VANETs e DTNs. Na secção 2.2 foi apresentado o conceito de VANETs. De acordo com a descrição dada, o impacto positivo que este tipo de redes pode ter na sociedade atual é muito relevante, levando a um novo conjunto de aplicações e serviços. Como resultado, a pesquisa e implementação deste tipo de redes já está a ser feito em muito países.

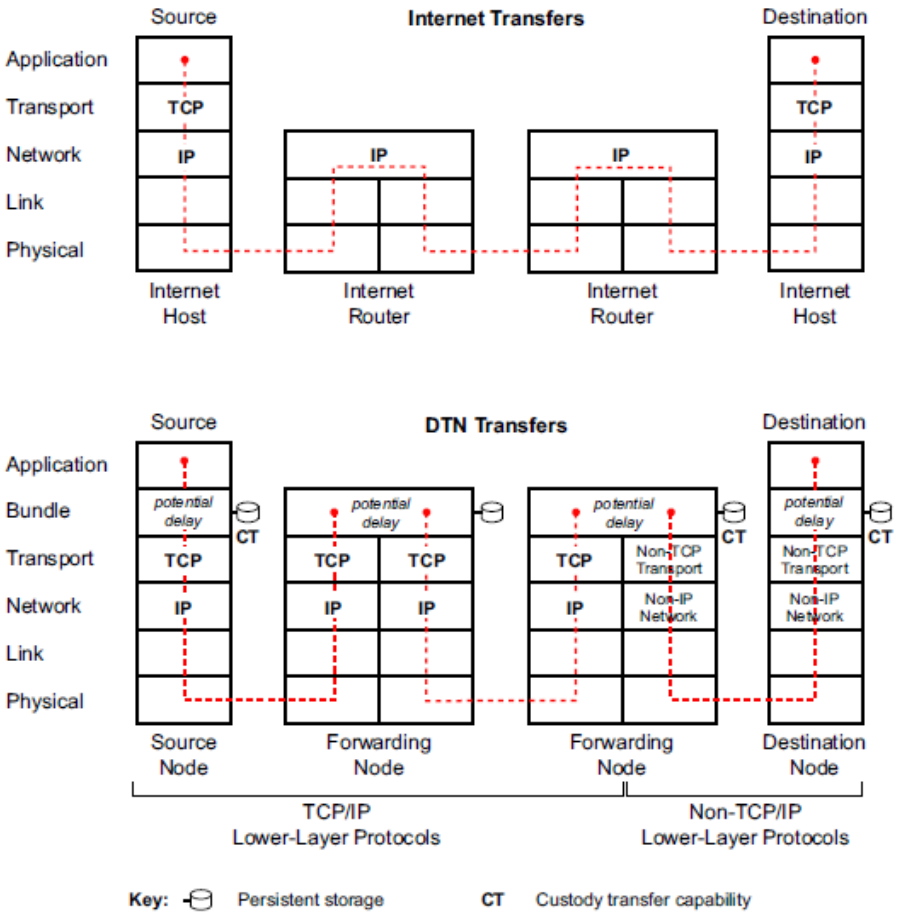


Figura 2.9: Diagrama de comunicação do Protocolo DTN vs. Diagrama de comunicação do Protocolo Internet [5].

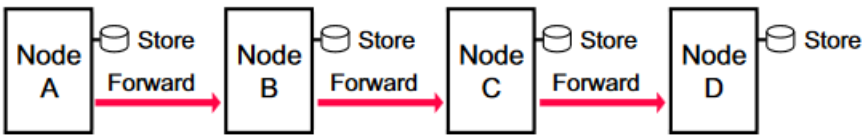


Figura 2.10: Mecanismo de Store-Carry-Forward usado nas DTNs [5]

A arquitetura de rede veicular é baseada em nós móveis (veículos) que interagem com a infraestrutura fixa para fornecer um conjunto de aplicações e serviços aos seus utilizadores. Uma rede veicular é caracterizada por mobilidade elevada e geralmente cobre uma área geográfica ampla. Assim, estas redes representam um ambiente com elevado potencial para fornecer e apoiar serviços de disseminação de conteúdos por meio de uma área de interesse significativa, utilizando veículos móveis para transportar os dados em disseminação para outros nós.

Na secção 2.3 o conceito de DTN foi apresentado, bem como as suas principais características e aplicações. O mecanismo SCF das redes DTN é uma das soluções para lidar com

as redes veiculares, uma vez que permite aos nós da rede o armazenamento das informações recebidas, o seu transporte, através da criação de uma cópia de informação, e a sua disseminação quando outros nós estão na área de cobertura do nó transmissor.

O principal objetivo deste trabalho é o desenvolvimento de estratégias de otimização do tráfego de dados e de controlo em mecanismos de distribuição de conteúdos para espalhar dados através de uma rede veicular. Assim, devido às condições adversas deste tipo de ambiente, uma DTN será usada para assegurar uma difusão confiável de dados não reais. A abordagem através de uma DTN leva a uma inovação na forma como os dados são distribuídos a partir de um servidor de conteúdos remoto e enviados para os nós móveis, usando veículos como mulas de dados para transportar dados.

Capítulo 3

Trabalhos Relacionados

3.1 Descrição do Capítulo

A disseminação de conteúdos nas VANETs recebeu grande atenção por parte dos investigadores, não só na área de segurança rodoviária, como também, na área de entretenimento (multimédia, redes sociais, informações turísticas, etc).

Nas VANETs, o conceito de distribuição de conteúdos assenta na seguinte ideia: as informações originais (vídeo específico disponível para *download*) são carregadas no servidor da Internet, pelo que os veículos que passam por uma infraestrutura fixa, RSU, ou por outro ponto de acesso, podem descarregar essas mesmas informações quando existe uma conectividade apropriada. Posteriormente, os veículos com essa informação podem divulgá-la aos veículos da sua proximidade, com os quais entram em contacto. Desta forma, o conteúdo é distribuído por toda a rede.

Para superar a curta duração de contacto com a infraestrutura, a tecnologia Peer-to-Peer (P2P) pode ajudar devido à sua característica nativa de particionar as informações e distribuí-las entre os nós da rede, através do *swarming* de ficheiros [31].

De forma a tornar a entrega de informação mais eficiente e com menos redundância, os nós da rede enviam pacotes de controlo antes de procederem ao envio de informação de dados. O pacote de controlo é enviado periodicamente em *broadcast* para manter a precisão sobre o conteúdo armazenado em relação a um determinado nó da rede. O seu tamanho depende da quantidade de informação que transporta.

Diversos trabalhos usam estratégias para disseminar conteúdo de forma eficiente nas VANETs. Estas estratégias requerem a introdução de pacotes de controlo para poderem descobrir os nós vizinhos e partilhar informação adequada. Neste capítulo vão ser apresentados vários trabalhos cujo objetivo é o controlo do excesso de *overhead* introduzido por estes pequenos pacotes de controlo, a fim de determinar qual abordagem poderia ser mais útil para diminuir a sobrecarga de uma rede veicular.

Como tal, este capítulo está organizado da seguinte forma:

- *secção 3.2*: *Mecanismos de Resumo de Informação para Distribuição de Conteúdos mais eficiente* - apresenta vários trabalhos cujo objetivo é o controlo do excesso de *overhead* introduzido pelos pacotes de controlo.

- *secção 3.3*: *Mecanismo de Distribuição de Conteúdos* - apresenta várias abordagens e protocolos de disseminação de conteúdos em VANETs.

- *secção 3.4* : *Considerações Finais* - apresenta o resumo deste capítulo.

3.2 Mecanismos de Resumo de Informação para Distribuição de Conteúdos mais eficiente

Os *Bloom filters* não são uma estrutura de dados recente e, mesmo no campo das comunicações veiculares, também não são uma novidade. *Marandi et al.* [32] realizaram um estudo extenso sobre o uso destes filtros em VANET.

Como o objetivo dos *bloom filters* é fornecer uma representação resumida dos elementos pertencentes a uma lista, as suas áreas de aplicação podem ser múltiplas. Segundo [13], os *bloom filters* podem ter as seguintes áreas de aplicação:

- Usados para resumir conteúdos em redes de *overlay* e *P2P*.
- Encaminhamento de recursos (ajuda na localização de um recurso) e pacotes (simplificam protocolos para o encaminhamento de pacotes, acelerando o seu processo).
- Usados para análise da informação, por exemplo, saber à priori se a informação pertence ou não à lista de informações consideradas como relevantes.

On-Demand Multicast Routing Protocol (ODMRP)

O ODMRP [33] é um protocolo para encaminhamento de tráfego em *multicast* para as MANETs. É baseado em malha, utiliza um conceito de grupo de encaminhamento (apenas um subconjunto de nós encaminha os pacotes em *multicast* através da técnica de *flooding*, cria dinamicamente rotas, e mantém a associação ao grupo *multicast*.

Mais tarde, *Yoneki et al* [34], sugeriram uma versão estendida do ODMRP para disseminação de dados através de subscrições baseadas em conteúdos. Os autores descrevem uma nova abordagem para o sistema de publicação/subscrição baseado em conteúdos para as MANET, e estenderam o *ODMRP* usando resumos agregados de subscrições baseadas em conteúdo, sob a forma de *bloom filters*, para a construção dinâmica de uma estrutura de disseminação de eventos. Posteriormente, estes resumos, são enviados para o grupo *multicast* mais apropriado.

Dadas as restrições das redes móveis, o envio de pacotes tem se ser feito o mais rápido possível. Portanto, esta ideia também poderia ser aplicável aos pacotes de controlo de uma rede veicular, diminuindo, consequentemente, o seu tamanho para posterior divulgação dos dados já armazenados.

Human Network (HUNET)

Também *Y. Zhao et al.* [7] propuseram a criação de uma rede humana HUNET através de dispositivos sem fios (smartphones). Nesta rede, os utilizadores são capazes de encaminhar mensagens específicas para outros utilizadores, de acordo com os seus interesses. Os dispositivos sem fios apenas encaminham mensagens para um *broker*. O *broker* é responsável pela correspondência entre o conteúdo e os utilizadores. Os *Bloom Filters* são usados para codificar os interesses dos utilizadores, tendo como principal vantagem o consumo de menos memória. Contudo, esta rede não é uma rede veicular. No entanto, este conceito de rede humana, sendo composta por utilizadores em constante movimento poderia ser aplicado a

uma rede VANET. A Figura 3.1 mostra um exemplo de uma HUNET. Cada pessoa está equipada com um dispositivo móvel diferente. Os padrões de contacto de cada pessoa são regidos pelos seus relacionamentos. Cada pessoa tem os seus próprios interesses, e é representada por: *um nome e par de interesses*. As mensagens são transportadas entre utilizadores através do mecanismo *store-carry-forwarding*. De igual forma, este mecanismo é usado para lidar com as redes veiculares.

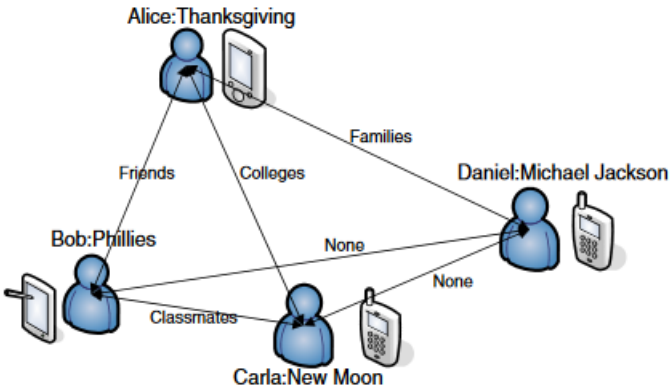


Figura 3.1: Exemplo de uma HUNET segundo [7].

Obfuscated Social Network Routing (OSNR)

Dado que a grande maioria das pessoas tem um dispositivo sem fios, *Iain Parris et al.* [35] observaram que o encaminhamento de mensagens poderia ser feito através das redes sociais. No entanto, este encaminhamento introduz preocupações em relação à privacidade de cada utilizador. Assim, os autores propuseram um protocolo de encaminhamento ofuscado através da rede social, designado de OSNR. Este protocolo usa *Bloom Filters* para proteger a lista de amigos de cada utilizador, codificando-a.

Generalized Bloom Filter (GBF)

Também *Laufer et al.* [36] introduziram um *Generalized Bloom Filter*, designado de GBF, para representar de forma segura um conjunto de aplicações distribuídas, como o *traceback* do IP, *web caching* e *peer-to-peer networks*.

Esta vantagem dos *Bloom Filters* vem fortalecer ainda mais o seu uso nas redes veiculares. A codificação, além de fornecer um tamanho mais pequeno dos pacote de controlo, também permite ocultar as informações contidas no filtro. Desta forma, o envio de dados sensíveis entre os nós da rede é protegido.

BlooGo

Angius et al [37] propuseram um algoritmo designado *BlooGo*, um algoritmo de *Gossip* que dissemina informações por toda a rede, com um número mínimo de transmissões. Este algoritmo permite que os nós enviem e recebam informações sem nenhum conhecimento prévio da rede.

Um nó decide se encaminha ou não pacotes de dados apenas com base nas informações sobre os nós vizinhos, codificadas num *Bloom filter*. Esta informação sobre os vizinhos diz

respeito ao conteúdo que cada um deles já tem armazenado num determinado momento, sendo análogo ao envio de pacotes de controlo. Se um nó recetor desta informação tiver todos os seus vizinhos com o mesmo conteúdo, sabe que não deve encaminhar pacotes de dados. Da mesma forma, se um vizinho não contiver essas informações, devem ser transmitidos pacotes de dados de forma a colmatar essa informação. Este mecanismo introduz alguma inteligência nos nós da rede, de forma a tomarem decisões sobre quais os pacotes de dados que devem enviar num determinado momento.

Esta abordagem poderá ser uma das aproximações do estudo desta dissertação, através do uso de *Bloom filter* nos pacotes de controlo. Assim, as informações são resumidas e os nós poderão enviar conteúdos com base nessas mesmas informações que recebem dos nós vizinhos.

Bloom Filter com tamanho fixo

Segundo Narkon et al. [8] é proposta uma solução usando o *Bloom filter* para criar *beacons* com tamanho fixo, de forma a tornar as redes mais escaláveis, sem afetar a transmissão dos dados.

Os autores referem duas propriedades do *Bloom filter*. A primeira é que o tamanho do *Bloom filter* não depende do número e do tamanho dos elementos inseridos. Esta propriedade ajuda na criação de *beacons* com tamanho fixo contendo uma estrutura de dados de tamanho variado. A segunda propriedade diz respeito ao conjunto de operações do *Bloom filter*. Este conjunto de operações pode ser implementado com operações *bitwise*, reduzindo a complexidade do algoritmo de $\emptyset(n^5)$ para $\emptyset(n)$.

Consideraram a análise de dois tipos de estruturas de *beacons* para usar o *Bloom filter* para melhorar a sua eficiência. A Figura 3.2 mostra o *Density-Aware Reliable Broadcasting Protocol*, designado DECA [38]. O seu *beacon* consiste no identificador do nó origem, o número de vizinhos a um salto de distância e a lista de pacotes identificados. Um único *Bloom filter* foi usado para representar esta lista de pacotes identificados.

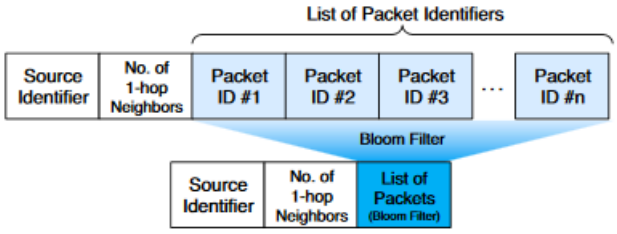


Figura 3.2: *Bloom filter* que representa a lista de pacotes identificados para a estrutura do *beacon* do protocolo DECA [8]

Por outro lado, os autores implementaram um outro protocolo de transmissão em *broadcast* confiável, usando o algoritmo de Wu e Li [39] no protocolo DECA, ao qual chamaram WLP. Este protocolo tem como objetivo representar protocolos que usam múltiplas listas de tamanho variado num único *beacon*, como mostra a Figura 3.3. Como resultado, o seu *beacon* representa o nó origem, uma lista de vizinhos com distância de um salto e uma outra lista de pacotes identificados.

O WLP é um exemplo de algoritmos cuja complexidade pode ser reduzida pelas propriedades do *Bloom filter*. Dois *Bloom filters* foram usados para representar cada lista. Depois da aplicação de um conjunto de operações bitwise, apenas um *Bloom filter* foi usado para representar ambas as listas. A Figura 3.4 mostra a redução efetuada.

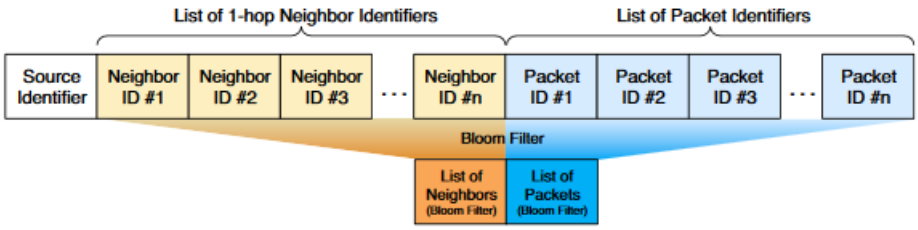


Figura 3.3: Dois *Bloom filters* que representam a lista de vizinhos com distância de um salto e a lista de pacotes identificados, respetivamente, para a estrutura do *beacon* do protocolo WLP implementado por [8]

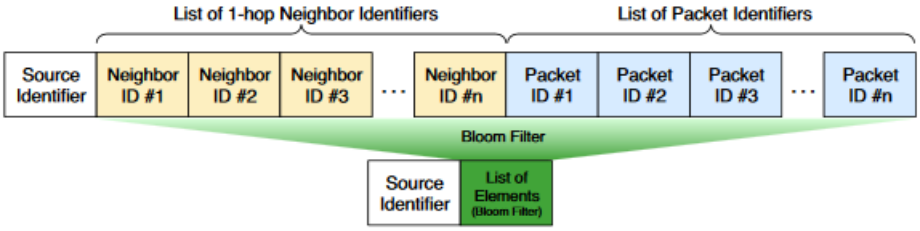


Figura 3.4: Resultado da aplicação das operações de *bitwise* para produzir um único *Bloom filters* representando ambas as listas do protocolo WLP implementado por [8]

Para reduzir o tamanho do *Bloom filter*, que reflete o tamanho do *beacon*, os autores estudaram vários ajustes entre o tamanho e a probabilidade de falsos positivos.

Os resultados a que chegaram indicam que a solução proposta reduz significativamente o *overhead* provocado pelos *beacons* atingindo apenas 4.8% da sua estrutura original.

Esta abordagem poderá ser, também, outra das aproximações do estudo desta dissertação, o uso de *Bloom filters* de tamanho fixo nos pacotes de controlo.

No entanto, o número de elementos (n = 142) que os autores inserem é relativamente baixo, o que leva a que o tamanho da *bit table (m)* do *bloom filter* seja mais baixo, e por isso, o tamanho do *beacon* conseguido será, também, mais baixo. Isto deve-se ao facto de que vão ser necessárias menos funções de *hash (k)* para calcular a posição de cada bit. Consequentemente, a probabilidade de falsos positivos também será mais baixa.

Apesar do cenário de simulação ser baseado na cidade de Bangkok, Tailândia, os dados foram gerados a partir de uma simulação de tráfego, denominada Simulação de Mobilidade Urbana (SUMO), o que não correspondem a dados reais.

PYRAMID

Segundo *Yu et al.* [9], a abordagem PYRAMID é uma abstração probabilística dos conteúdos armazenados nos veículos. Os autores utilizam um conjunto de estruturas de dados probabilísticas para abstrair e aproximar de forma eficiente o conteúdo para diferentes tipos de granularidade. Particularmente, *sketchs* de granularidade grosseira (*FM Sketch, AMS Sketch, Minwise Sketch* e *Z-Smallest Sketch*) estimam o número de pacotes do conteúdo, bem como o nível de similaridade do conteúdo entre dois veículos da rede. Resumos de granularidade fina (*Index, Bitmap* e *Bloom Filter*) são usados para teste de associação.

O PYRAMID desenvolveu dois mecanismos, a prioritização de tarefas e a reconciliação

de conteúdos, dois processos críticos em aplicações veiculares P2P. A prioritização de tarefas (primeira fase) diz respeito à identificação apropriada do veículo mais valioso, isto é, que contém um maior número de conteúdos que está em falta noutro veículo, entre os diversos veículos da vizinhança. Para proceder a essa identificação são usados *sketchs* para estimar a contribuição do valor de utilidade de potenciais nós parceiros de transação. A reconciliação de conteúdo (segunda fase) tem como objetivo informar o nó sobre quais os conteúdos a transferir, sem que haja redundância. Para isso, os resumos ajudam a determinar a parte do conteúdo complementar entre dois veículos.

Como resultado, é feita a disseminação do conteúdo entre o veículo que envia e o veículo que o elegeu. A Figura 3.5 sumariza a abordagem PYRAMID.

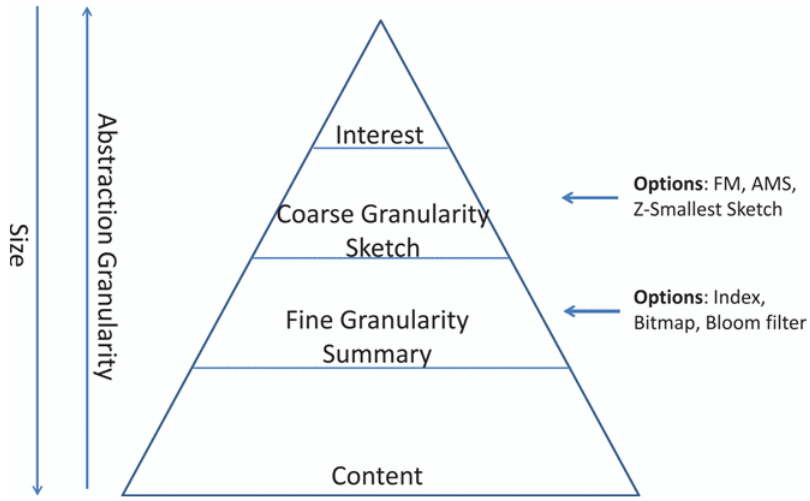


Figura 3.5: PYRAMID - abstração de informações multi-camada proposta por [9]

Os autores avaliaram a abordagem PYRAMID em ambientes do mundo real usando uma frota de quatro veículos equipados com tecnologias DSRC. Demonstraram experimentalmente que, para diversos cenários, a prioritização de tarefas melhora o valor de utilidade nas trocas de conteúdos em 20%-30% e a reconciliação de conteúdo inteligente por meio de testes de associação melhora o rendimento efetivo em pelo menos 25%.

A abordagem PYRAMID recebeu também especial atenção por parte do estudo desta dissertação, uma vez que faz ênfase ao uso de várias aproximações, com o objetivo de reduzir o *overhead* provocado pelos pacotes de controlo na rede. O uso de um *bit array* de tamanho fixo, para representar a informação sobre o conteúdo armazenado em cada nó vizinho, poderá ser outra das aproximações de estudo.

Hierarchical Bloom-Filter based Routing (HBFR)

Também *Yu et al.* [40] introduziram um algoritmo pro-ativo de divulgação e descoberta de conteúdo, designado *Hierarchical Bloom-Filter based Routing* (HBFR), para lidar com mobilidade, grande quantidade de população e entrega de conteúdo valioso. Os resultados da simulação, a que os autores chegaram, mostram que o HBFR supera as abordagens de *flooding* em termos do volume de tráfego induzido, ao recuperar dados populares.

Este estudo vem ainda reforçar mais uso de *bloom filters* para reduzir o *overhead* introduzido na rede.

Greedy Perimeter Stateless Routing (GPSR)

Também em[41] se propõe uma extensão ao protocolo GPSR, em que usam o *bloom filter* para filtragem do conteúdos das mensagens recebidas de outros veículos da rede, de forma a eliminarem a informação que não é necessária. O GPSR é um protocolo de encaminhamento que depende da posição geográfica dos nós, apropriado para as VANET. Neste protocolo todos os nós transmitem as suas informações de localização periodicamente para os nós vizinhos. A informação recebida pelos nós vizinhos é armazenada numa tabela com informações dos vizinhos. O nó transmissor da informação seleciona o próximo nó de encaminhamento, de acordo com as informações da localização do vizinho e o local de destino da informação. A fim de promover eficazmente o pacote desejado, o GPSR utiliza a informação do vizinho mais próximo do destino. Cada nó determina também o número de nós vizinhos e as suas velocidades médias e, em seguida, transmite essas informações para outros nós vizinhos. Informações como o clima, condições da estrada, a interseção de autoestradas e o sinal do veículo de emergência, também são coletados pelos veículos.

Assim, os autores usam o *bloom filter* para filtrar o conteúdo das mensagens, e apenas a informação relativa ao controlo do veículo é propagada para os outros nós. A restante informação é eliminada. O algoritmo de filtragem usado é baseado na estrutura de descoberta de conteúdos ponto a ponto.

Contudo, o objetivo deste trabalho vai em contrário ao objetivo desta dissertação, uma vez que o *bloom filter* é usado, não para resumir o conjunto da informação enviado nas mensagens, mas sim para filtragem do conteúdo das mensagens recebidas de outros veículos da rede.

3.3 Mecanismos de Distribuição de Conteúdo

3.3.1 Macanismos Peer-to-Peer

Swarming Protocol For Vehicular Ad-Hoc Wireless Networks (SPAWN)

O *Swarming Protocol For Vehicular Ad-Hoc Wireless Networks*, designado de SPAWN [10], segue a mesma estrutura de *P2P swarming protocol*. Os nós descarregam partes de um ficheiro da infraestrutura e partilham essas mesmas partes entre os seus nós vizinhos.

Este processo está ilustrado na Figura 3.6

Como mostra a Figura 3.6 (1), um carro chega ao alcance de uma *gateway*, (2) inicia o *download* de um ficheiro (3) faz o *download* de uma parte do ficheiro. (4) Depois de sair da área de cobertura da *gateway*, (5) usa o contacto oportunístico com os seus vizinhos sobre a disponibilidade do conteúdos e (6) troca partes do ficheiro, obtendo assim, uma porção maior do ficheiro, em vez de esperar pelo próximo contacto de uma *gateway*, para retomar o ficheiro.

CarTorrent

Mais tarde, por extensão ao protocolo SPAWN, *Lee et al.* [42] propuseram o *CarTorrent*. *CarTorrent* é um protocolo de descarregamento de ficheiros baseado no *BitTorrent*, desenvolvido para as redes veiculares. Os clientes *CarTorrent* usam o contacto oportunístico, limitado a uma distância de k saltos para disseminar as partes do ficheiro que tem disponíveis. As mensagens de contacto oportunístico são propagadas até k saltos desde a origem. Isto permite que outros nós da rede recolham informações sobre a disponibilidade das partes do ficheiro, bem como sobre a topologia local da rede. As informações de topologia e disponibilidade são usadas para selecionar a próxima parte a ser solicitada a seguir. Por exemplo, se o nó A e

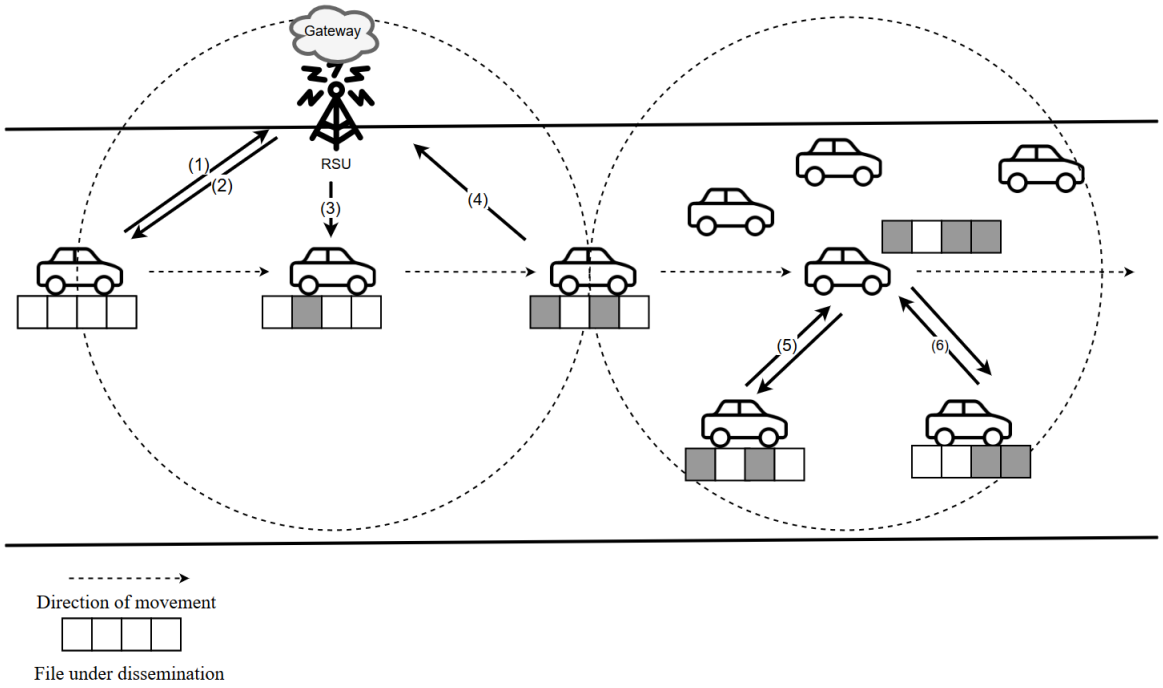


Figura 3.6: Evolução do descarregamento de um ficheiro nos nós da rede, usando o protocolo SPAWN [10].

---

o nó B possuem uma parte mais rara que o nó C deseja, e A está a uma distância menor que B, então o pedido é enviado para A. Assim, no *CarTorrent*, os veículos são capazes de determinar qual é a parte mais rara do ficheiro na rede e o nó mais próximo que a armazena.

RoadCast

Zhang et al. [43] propuseram uma estratégia de distribuição de conteúdos P2P para as VANETs, designada *RoadCast*. Esta estratégia baseia-se no aumento da probabilidade em obter o conteúdo mais popular na rede. Para isso, os nós anunciam o quão popular é determinado conteúdo na rede. São ainda implementados algoritmos para assegurar que o conteúdo mais popular está presente num maior número de nós.

Mobility-Centric Data Dissemination (MDDV)

De acordo com *Wu et al.* [44] o *Mobility-Centric Data Dissemination* (MDDV) combina o encaminhamento oportunístico baseado na trajetória dos nós da rede com o encaminhamento geográfico baseado na posição dos nós. Os pacotes são encaminhados através de uma trajetória bem definida, em que os nós intermediários armazenam estes pacotes num *buffer*, enviando-os quando surge um encontro oportuno. O MDDV impõe regras de transmissão aos nós da rede, dizendo se um nó pode ou não transmitir pacotes, com vista a diminuir o congestionamento na rede.

O algoritmo usado no MDDV encaminha os pacotes com base na posição geográfica dos nós da rede, em vez do seu conteúdo, o que vai contra o objetivo pretendido desta Dissertação.

REceiver-based solution with video transmission DECoupled (REDEC)

Rezende et al. [45] propuseram REDEC, um mecanismo de distribuição de conteúdo pensado especificamente para *streaming* de vídeo. REDEC considera que um veículo pode ter um dos quatro estados diferentes: *non-relay, scheduled, relay* ou *scheduled relay*. O estado *non-relay* não transmite pacotes, apenas está interessado em recebê-los. Um nó de *relay* transmite em *broadcast* pacotes e recebe pacotes. Um nó de *scheduled/scheduled relay* recebe um pacote de controlo e decide se irá transmitir em *broadcast* pacotes de dados ou não. Enquanto o nó se encontra no processo de decisão encaminha pacotes; só quando a decisão de não transmitir pacotes for tomada é que pára de transmitir.

O objetivo dos quatro estados mencionados é a diminuição da sobrecarga na rede, limitando o número de nós capazes de enviar pacotes de controlo. Dado que o objetivo desta Dissertação não é limitar o envio de pacotes de controlo, mas sim diminuir o seu tamanho.O *REDEC* não é uma estratégia preferida para distribuição de conteúdos.

Distributed Vehicular Broadcast (DV-CAST)

O *DV-CAST* [46] usa mensagens de Hello periódicas, recebidas de vizinhos à distância de um salto, para construir a topologia da rede. Com a construção da topologia da rede é possível a retransmissão do pacote a ser disseminado. O DV-CAST está adaptado quer para cenários densos ou mais esparsos. Em cenários onde a topologia da rede está bem conectada, aplica o algoritmo de supressão de *broadcast* [47]. Para cenários mais esparsos, o mecanismo *store-carry-forward* é usado como solução. O seu desempenho é dependente da frequência com que as mensagens de Hello são enviadas, sendo muito difícil de estabelecer um valor ótimo. Além disso, apenas está adaptado para cenários de autoestradas, e não há partilha de informações sobre se tal pacote já foi ou não recebido, pelo que vai contra o objetivo desta Dissertação, cujo principal foco é o cenário urbano.

Adaptive approach for Information Dissemination (AID)

AID [48] é um mecanismo descentralizado e adaptativo para disseminação de dados nas VANETs. Neste mecanismo, os nós da rede encaminham ou não um pacote consoante o número de vezes que receberam o mesmo pacote, para um dado período de tempo. Em cenários densos, os nós da rede podem suprimir o envio de um pacote, dado que já foi encaminhado por diversos nós, levando à diminuição do problema de *broadcast storm*. Contudo, o AID não tem em conta a fragmentação temporal da rede e problemas de partição da rede. Além disso, não usa pacotes de controlo para informar quais os recursos de cada nó.

DRIFT

O *DRIFT* [49] está adaptado apenas para cenários de autoestrada, promovendo apenas a comunicação V2V, não necessitando do suporte de infraestruturas. DRIFT usa uma zona preferencial para eliminar o problema de *broadcast storm*. Assim, *DRIFT* dá uma maior preferência de retransmissão para os veículos que estão em mobilidade na mesma direção do veículo originário da informação. Contudo, não lida com o problema de partição da rede, nem está adaptado para cenários urbanos. Além disso, os nós da rede não informam sobre qual o conteúdo que cada um deles possui, indo em contraposição ao objetivo desta Dissertação.

3.3.2 Protocolos de Disseminação de Informação em VANETs

A disseminação de informação nas redes VANETs assenta, sobretudo, em protocolos de transmissão em *broadcast*, uma vez que a informação é de interesse de todos os veículos que estão na rede, e não de apenas um em específico, como acontece na transmissão em *unicast*. Outra das vantagens da transmissão em *broadcast* é que os veículos da rede não necessitam de saber a rota nem o endereço de um destino em específico, eliminando a complexidade inerente à descoberta da rota, resolução de endereços e gestão da topologia, que constituem uma das maiores dificuldades nas redes VANETs. Como tal, é mais apropriado usar uma abordagem de transmissão em *broadcast* do que uma abordagem que use transmissão em *unicast*.

Segundo *Panichpapiboon et al.*, [11] os protocolos de transmissão de informação em *broadcast* estão divididos em duas classes principais: (1) *Multi-hop Broadcast* e (2) *Single-hop Broadcast* e são apresentados na Figura 3.7. A principal diferença entre estas duas classes reside na maneira como a informação é espalhada na rede. Nos protocolos de *multi-hop broadcast*, a informação propaga-se na rede através da técnica de *flooding* ("inundam"a rede), isto é, todos os veículos enviam pacotes para todos os seus vizinhos. Consequentemente, todos os vizinhos ao receberam pacotes enviam-nos, por sua vez, para todos os seus vizinhos, e assim sucessivamente. Como resultado, os pacotes serão capazes de se propagar desde a sua origem para os outros veículos mais distantes, alcançando todos os nós pertencentes à rede.

Pelo contrário, nos protocolos de *single-hop broadcast*, quando um veículo recebe a informação, mantém-na consigo mesmo, atualizando apenas a sua base de dados, ou seja, não usa a técnica de *flooding*. Além disso, cada veículo seleciona a informação mais relevante da sua base de dados para transmitir periodicamente. Essa informação será transferida para outros nós da sua vizinhança à distância de um salto nos próximos ciclos de transmissão. Como tal, os protocolos de *single-hop broadcast* dependem muito da mobilidade de nós na divulgação das informações.

As subsecções seguintes apresentam resumidamente cada um dos subgrupos em que se dividem.

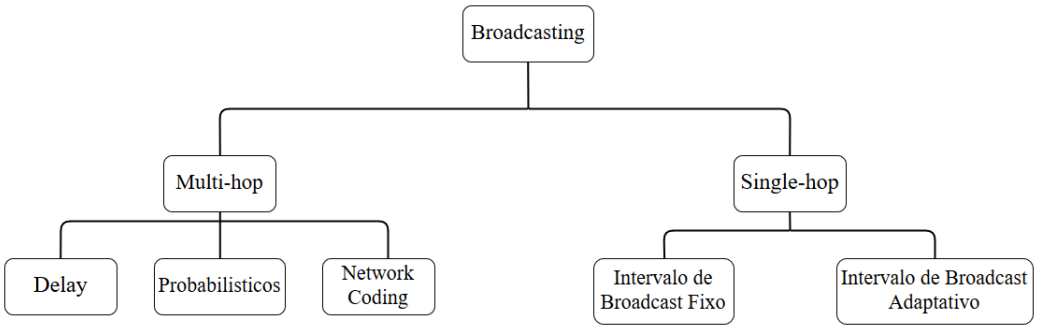


Figura 3.7: Classificação de Protocolos de disseminação de Informação, baseado em [11].

3.3.2.1 Protocolos de *Multi-hop Broadcast*

A razão para existirem três categorias de protocolos *Multi-hop* reside no facto de que a técnica de puro *flooding* da informação na rede se tornar ineficiente com a escabilidade da rede, e a quantidade de pacotes redundantes que circulam na rede aumentar significativamente, como se poderia esperar. Para resolver estes dois problemas, uma solução possível é reduzir

o número de retransmissões redundantes de pacotes. Esta solução é conseguida através da seleção de apenas alguns nós para retransmissão de pacotes. Seguidamente são apresentadas abordagens existentes usadas para reduzir o número de transmissões de pacotes.

3.3.2.1.1 Mecanismos baseados em atrasos Estas abordagens consistem, sobretudo, em dar um tempo de atraso (de espera) diferente na retransmissão dos pacotes, a cada um dos nós recetores. Assim, o nó com um tempo de espera mais curto tem a maior probabilidade de retransmitir o pacote.

Para evitar a redundância na retransmissão de pacotes, sempre que um nó sabe que um pacote foi retransmitido, aborta o seu processo de espera de retransmissão para esse pacote. O cálculo do tempo de espera de cada veículo corresponde a uma função da distância entre o veículo e o transmissor (entenda-se por transmissor, a origem da informação). Tipicamente, ao veículo mais longe é-lhe dado o tempo de espera mais curto, o que significa que é implicitamente selecionado como o próximo nó retransmissor.

Os mecanismos baseados em atrasos são *Urban Multi-hop Broascast* (UMB), *Smart Broadcast* (SB), *Efficient Directional Broadcast* (EDB), *Slotted 1-Persistence Broadcasting*, *Multi-hop Veicular Broadcast* (MHVB), *Multi-hop Veicular Broadcast* (MHVB), e por último *Link-based Distributed Multi-hop Broadcast* (LDMB).

Urban Multi-hop Broadcast (UMB)

O Protocolo UMB [50] foi proposto para resolver problemas de fiabilidade na transmissão, bem como de *broadcast storm*. UMB divide a estrada em pequenos segmentos, dentro da área de cobertura do transmissor, dando a prioridade de retransmissão aos nós que pertencem ao segmento mais distante.

Existem dois tipos de encaminhamento de pacotes: *broadcast* direcional e de interseção. O *broadcast* direcional funciona da seguinte forma: quando um nó tem um pacote para enviar, primeiro transmite um pacote de controlo, designado *Request-To-Broadcast* (RTB), que engloba a sua posição e a direção de propagação do pacote. Todos os nós que se encontram na área de transmissão do emissor recebem o pacote RTB, e cada um deles começa a transmitir um sinal de interferência, chamado *blackburst*, por um período de tempo especifico. A duração do *blackburst* corresponde a uma função da distância entre o veículo que recebe o pacote RTB e o emissor, apresentada em [50].

Depois de um nó transmitir o *blackburst*, coloca-se à escuta do canal. Se deteta que o canal está ocupado (outros nós estão a transmitir os *blackbursts*), o veículo não faz nada, e delega o dever de retransmissão para outros nós que ainda estão a transmitir os *blackbursts*. Por outro lado, se deteta que o canal está desimpedido, transmite um pacote de controlo designado *Clear-To-Broadcast* para o nó que iniciou a transmissão do pacote RTB, sendo designado o próximo nó retransmissor do pacote.

Quando o nó que iniciou o pacote RTB recebe o pacote CTB, transmite o pacote de dados. De seguida, quando o nó, designado como o próximo nó, recebe o pacote de dados (DATA), envia um pacote ACK ao transmissor do pacote de dados. Se o pacote ACK não for recebido pelo nó transmissor durante um tempo específico, todo o processo RTB-CTB-DATA-ACK é iniciado novamente.

No entanto, este protocolo não resolve todas as colisões, sendo possível que mais do que um nó, no mesmo segmento de estrada, possa transmitir ao mesmo tempo pacotes CTB.

Este problema é agravado pelo aumento da densidade de nós. O encaminhamento através de *broadcast* de interseção é usado para a transmissão de um pacote aquando da interseção da estrada principal com outras estradas (cruzamentos). Para isso, um nó terá de estar posicionado na interseção para encaminhar o pacote para as outras estradas.

Smart Broadcast (SB)

O protocolo SB [51] é proposto para melhorar a limitação do protocolo anterior (UMB). O UMB é ineficiente quando o nó, designado de próximo nó retransmissor, tem de esperar um maior tempo para transmitir o pacote CTB. Tal acontece porque a maior duração do *blackburst* é atribuída ao próximo veículo de retransmissão. O SB resolve este problema, atribuindo ao próximo veículo de retransmissão o menor atraso de espera.

Este protocolo é melhor do que o protocolo UMB, em termos de latência, mesmo para grandes mudanças na densidade de nós da rede.

Efficient Directional Broadcast (EDB)

O protocolo EDB [52] opera de forma semelhante relativamente aos dois protocolos descritos anteriormente. Contudo, os pacotes de controlo RTB e CTB não são usados neste protocolo. Explora também o uso de antenas direcionais, pelo que os veículos têm de estar equipados com duas antenas direcionais, cada uma com 30 graus de largura de feixe.

No *broadcast* direcional no segmento de estrada, um veículo (de onde a informação tem origem) transmite um pacote de dados e os veículos a jusante retransmitem-no para outros nós da rede.

Para reduzir o número de pacotes de retransmissão redundantes, o EDB atribui um tempo de espera diferente, antes da retransmissão, para cada um dos veículos dentro da área de cobertura do transmissor. O tempo de espera é uma função da distância entre o veículo e o transmissor, apresentada em [52]. Depois do tempo de espera expirar, o nó transmite imediatamente um pacote ACK para informar os seus nós vizinhos que não é necessário desempenharem a tarefa de retransmissão do pacote de dados. Depois de transmitir um pacote ACK, o nó pode iniciar a retransmissão do pacote de dados. O veículo mantém periodicamente a retransmissão do pacote, se nenhum outro veículo encaminhar o pacote, dentro de um intervalo máximo.

Também no *broadcast* de interseção, um nó deve ser posicionado numa interseção (cruzamento) para transmitir os pacotes para as outras direções da estrada.

O EDB consegue reduzir as colisões de pacotes de dados, através do envio de um pacote ACK para os seus nós vizinhos.

Slotted 1-Persistence Broadcasting

O encaminhamento de pacotes efetuado no protocolo *Slotted 1-Persistence Broadcasting* [53], é também, similar aos protocolos anteriormente referidos. Neste protocolo, quando um nó recebe um pacote de dados, retransmite-o, de acordo com um intervalo de tempo definido por uma função da distância entre o nó e o emissor, apresentada em [53], e se nenhum outro nó o transmitiu.

No entanto, este protocolo não resolve as colisões de pacotes de dados, sendo possível que mais do que um nó transmita, ao mesmo tempo, o pacote no mesmo intervalo de tempo definido.

Multi-hop Veicular Broadcast (MHVB)

Como outros protocolos desta categoria, o protocolo MHVB [54], quando recebe um pacote, calcula o tempo de espera antes de retransmitir o pacote, baseado na distância entre ele próprio e o nó emissor. Um tempo curto de espera vai ser atribuído ao nó cuja distância é maior. Quando o tempo de espera acaba, o nó retransmite então o pacote. Se o nó "ouvir"uma duplicação, todo o processo é cancelado.

O MHVB deteta o congestionamento do tráfego. Como tal, quando existe congestionamento de tráfego, a densidade de nós aumenta. Como resultado, o intervalo de retransmissão é extendido. O mecanismo de deteção de congestionamento de tráfego funciona da seguinte forma: cada nó usa o número de vizinhos e a sua velocidade como indicação de congestionamento; Quando um veículo deteta que o número de vizinhos é superior a um certo valor limite, e que a sua velocidade é inferior a um valor limite, pode ser indicação de congestionamento de tráfego.

Link-based Distributed Multi-hop Broadcast (LDMB)

O protocolo LDMB [55] atribui o tempo de espera baseado em ligações de qualidade. Aquando do cálculo do tempo de espera, LDMB não só considera apenas a distância entre o emissor e o recetor, como também considera outros fatores como a densidade dos nós, área de transmissão, e a taxa de transmissão dos pacotes. Contudo, o seu desempenho não é melhor em termos de taxa de entrega, do que os protocolos que apenas têm em consideração a distância entre o transmissor e o recetor.

Apesar de todos estes protocolos mencionados nesta subsecção terem um mecanismo de resolução do aumento da quantidade de pacotes redundantes na rede, não consideram o uso de pacotes de controlo para informar os nós vizinhos sobre quais os recursos que têm disponíveis.

3.3.2.1.2 Mecanismos Probabilísticas Estes mecanismos baseiam-se em atribuir diferentes probabilidades de retransmissão (ou probabilidade de encaminhamento) a cada nó da rede. Como tal, cada veículo retransmite um pacote, de acordo com a probabilidade de retransmissão atribuída.

Um dos grandes desafios destes mecanismos é determinar a função de atribuição de probabilidade ótima. Enquanto que um simples protocolo usa um valor pré-definido de probabilidade de encaminhamento, outros protocolos mais sofisticados deixam cada nó ajustar a sua probabilidade dinamicamente, baseado em fatores como a localização do nó e a densidade da rede.

Alguns exemplos de mecanismos probabilísticos são *Weighted p-Persistence*, *Optimized Adaptative Probabilistic Broadcast* (OAPB), *AutoCast* e *Irresponsable Forwarding*.

Weighted p-Persistence

Neste protocolo [53], quando um veículo recebe um pacote pela primeira vez, calcula a sua probabilidade de retransmissão baseada na distância entre ele próprio e o transmissor. Esta distância pode ser obtida comparando a sua posição com a posição do transmissor, especificada no pacote, dando origem à função 3.1:

$$pij = \frac{Dij}{R} \tag{3.1}$$

onde *Dij* é a distância entre o transmissor i e o nó j, R é a área de transmissão. Como resultado, o nó que estiver a uma maior distância do transmissor obtém a maior probabilidade de retransmissão.

No entanto, este protocolo não tem em consideração a densidade da rede.

Protocolo Optimized Adaptative Probabilistic Broadcast (OAPB)

No OAPB [56], a probabilidade de encaminhamento é calculada a partir da densidade local do veículo (em termos do número de vizinhos), sendo obtida através de pacotes HELLO.

Quando um veículo recebe um pacote, calcula a sua probabilidade de encaminhamento com base na equação 3.2

$$p = \frac{P0 + P1 + P2}{3} \tag{3.2}$$

onde P0, P1 e P2 são funções do número de vizinhos de um salto, do número de vizinhos de dois saltos, e um conjunto de vizinhos de dois saltos que só podem ser alcançados através de um vizinho de um salto em particular.

Para reduzir o número de retransmissões de nós, cada nó retransmissor com a mesma probabilidade de encaminhamento p, atribui um atraso diferente, calculado por:

$$\Delta(t) = \Delta(t)max \times (1 - p) + \delta \tag{3.3}$$

onde $\Delta(t)max$ é o atraso de tempo máximo e δ é um valor aleatório variável na ordem dos milissegundos.

Protocolo *AutoCast*

No protocolo *Autocast* [57], a probabilidade de retransmissão é calculada a partir do número de vizinhos em torno do veículo, através da equação 3.4:

$$p = \frac{2}{Nh \times 0.4} \tag{3.4}$$

onde Nh é o número de vizinhos à distância de um salto, para $Nh \geq 5$.

No entanto, devido à natureza da técnica probabilística de *flooding*, um pacote pode nem sempre ser capaz de alcançar os veículos mais distantes, porque alguns veículos podem decidir não encaminhá-lo. Para aumentar a cobertura e o alcance, neste protocolo, um pacote é também retransmitido periodicamente, com base na equação 3.5:

$$t = \frac{Nh}{\alpha} \tag{3.5}$$

onde α é uma constante que especifica o número desejado de pacotes transmitidos em *broadcast* por segundo.

Quando comparado com outros protocolos, o *Autocast* tem uma maior taxa de entrega e rapidez de disseminação.

Irresponsable Forwarding (IF)

O protocolo IF [58] atribui a probabilidade de encaminhamento com base na distância entre o veículo e o emissor, assim como, a densidade de veículos. Assim, sempre que um nó recebe um pacote calcula a probabilidade de encaminhamento com base na equação 3.6:

$$p = e^{-\frac{\rho s(z-d)}{c}} \tag{3.6}$$

onde ρs é a densidade de nós, z é a área de transmissão, e d é a distancia entre o veiculo e o transmissor, e $c \geq 1$ é um coeficiente que pode ser selecionado para moldar a probabilidade de retransmissão p. Assim, quanto maior for o valor de c, maior será a probabilidade de retransmissão. O número de retransmissões pode ser controlado ajustando este coeficiente.

Quanto maior for a distância entre o nó e o transmissor, maior será a probabilidade de retransmissão. Quando a rede se torna mais densa, a probabilidade de retransmissão diminui, o que é um comportamento desejado.

De igual forma, todos os protocolos mencionados nesta subsecção procuram resolver o problema de aumento de escabilidade, assim como o aumento do congestionamento provocado pelo aumento de pacotes redundantes na rede. No entanto, também não consideram o uso de pacotes de controlo, para informar os nós vizinhos sobre quais os recursos que têm disponíveis.

3.3.2.1.3 Mecanismos baseados em *Network Coding* A diferença entre *Network Coding* e a abordagem tradicional de transporte é apresentada na Figura 3.8.

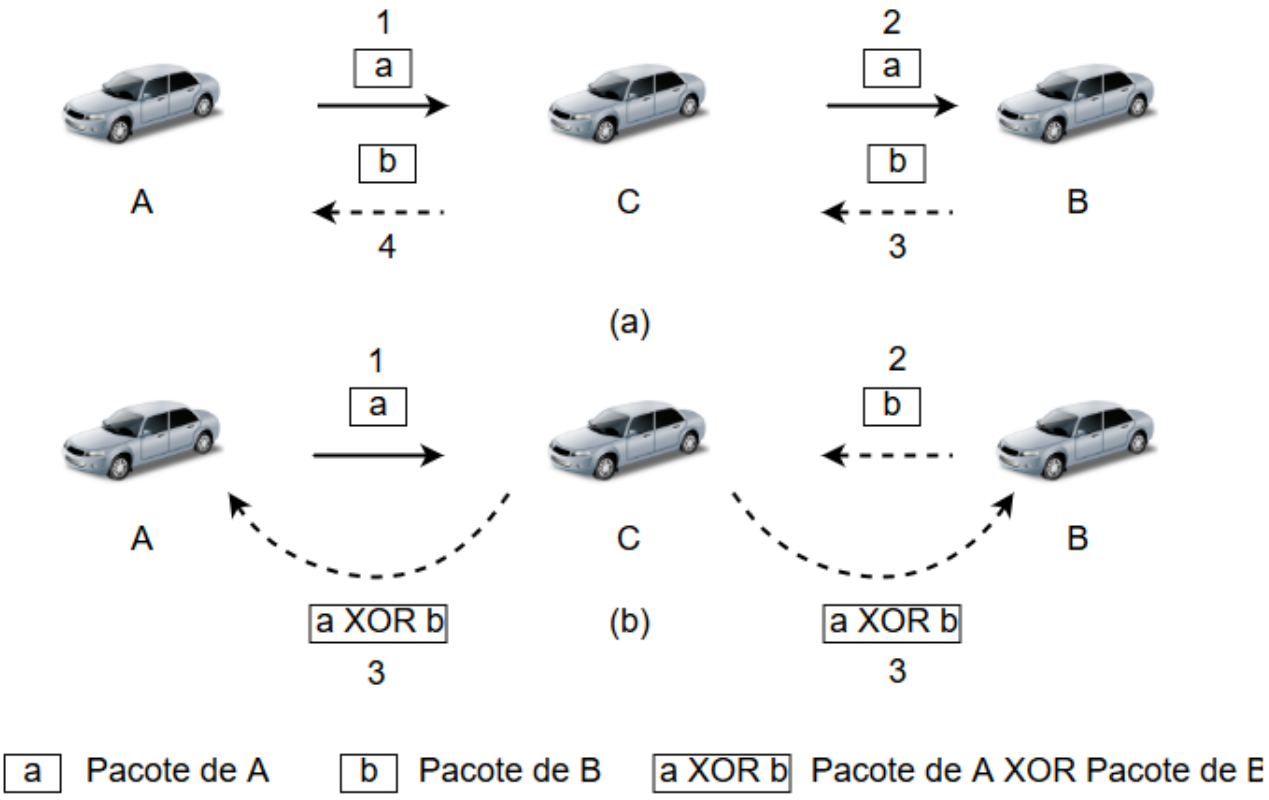


Figura 3.8: (a) Exemplo de transmissão tradicional (b) Exemplo de transmissão através de *Network Coding* [11].

Considera-se uma rede veicular simples, como mostra a Figura 3.8 (a), onde o nó C é um nó intermediário entre o nó A e o nó B. Como tal, o nó A e B não estão diretamente ligados. Suponha-se que A quer enviar um pacote para B, e B quer enviar um pacote para A. Na abordagem de transmissão tradicional, A necessita de enviar o pacote para C, e deixar que

C encaminhe o pacote para B. O mesmo acontece para B, que envia um pacote para A via C. O número de transmissões requeridas para completar as trocas de pacotes, neste caso, são quatro. Na transmissão usando *Network Coding*, a troca de pacotes entre A e B pode ser realizada com um número menor de transmissões, como mostra a Figura 3.8 (b). Primeiro A transmite o pacote para C. Segundo, B transmite o pacote para C. Depois de C receber ambos os pacotes de A e B, cria um pacote codificado, através da realização da operação XOR sobre os pacotes recebidos de A e B, e envia o resultado. Finalmente, cada um dos nós A e B podem descodificar o pacote recebido de C, através da realização da operação XOR sobre o pacote recebido e o seu próprio pacote (pacote com destino final A, e B, respetivamente). De notar que este processo apenas requer 3 transmissões.

Como resultado, a disseminação através de *Network Coding* requer um número menor de transmissões, usando a largura de banda de forma mais eficiente. O principal desafio está em como aplicar este conceito nas redes VANET.

Os protocolos baseados em *network coding* são *CODEB*, *Efficient Broadcast Using Network Coding and Directional Antennas* (EBCD) e *DiffCode*.

CODEB

O protocolo CODEB [59] estende os conceitos e técnicas usadas no protocolo COPE [60], para englobar cenários de *broadcast* nas VANETs. Baseia-se na escuta oportunista, pelo que cada nó adiciona ao seu *buffer*, por um determinado tempo limite, todos os pacotes que "ouve"da rede. Além disso, cada nó transmite periodicamente uma lista dos seus vizinhos à distância de um salto, o que permite a cada nó construir uma mapa da rede de transmissão.

CODEB depende da codificação oportunistica, determinando se um nó pode explorar oportunidades de codificação para transmitir pacotes codificados para os seus vizinhos. De notar que a codificação oportunistica para transmissão em *broadcast* é bem diferente da codificação para *unicast*. No encaminhamento *unicast*, somente o nó à distância de um salto recebe um determinado pacote, enquanto que no encaminhamento em *broadcast* todos os vizinhos recebem o pacote. Isto adiciona outro nível de complexidade, pois um nó deve garantir que todos os seus vizinhos possuem todas as informações necessárias para que consigam descodificar o pacote.

Cada nó cria uma lista de encaminhamento com um subconjunto de vizinhos que devem encaminhar o pacote deterministicamente. Esta lista contém o número mínimo de nós de difusão, de forma a que todos os nós da sua vizinhança, à distância de dois saltos, sejam cobertos. No entanto, um nó desta lista pode decidir não encaminhar o pacote, se todos os seus vizinhos já receberam o pacote.

Efficient Broadcast Using Network Coding and Directional Antennas (EBCD)

O protocolo EBCD [55] combina *Network Coding* com as antenas direcionais. De forma similar ao protocolo *CODEB*, também determina um subconjunto de nós vizinhos que vão encaminhar o pacote, de forma determinística, usando um algoritmo designado *Dynamic Directional Connected Dominating Set*. Este algoritmo é responsável por construir uma rede virtual direcional, onde cada nó determina quer o estado de encaminhamento, quer os setores de antena nos quais os pacotes serão transmitidos.

Outra diferença principal entre este protocolo e o *CODEB* é que, no EBCD, o *Network coding* é aplicado em cada setor das antenas direcionais, em torno do nó, em vez de ser omnidirecional. Assim, o uso de antenas direcionais e o *Network coding* fornecem uma melhoria

significativa em termos de número de transmissões, perante uma abordagem que usa apenas *Network coding* e uma abordagem que não usa nenhuma das duas.

DiffCode

O protocolo *DiffCode* [61] seleciona os nós responsáveis pelo encaminhamento dos pacotes, de forma determinística, usando um algoritmo baseado em Multi-Point Relay (MPR).

Um MPR de um nó é definido como um conjunto dos seus vizinhos à distância de um salto, que cobre a sua vizinhança à distância de dois saltos. No *DiffCode* cada nó na rede codifica e difunde apenas os pacotes que são recebidos dos nós que o selecionam como sendo o seu MPR.

Ao contrário do protocolo CODEB, que limita a oportunidade de codificação (todos os seus vizinhos têm de ser capazes de descodificar um pacote), o *DiffCode* permite aos nós da rede armazenarem pacotes em *buffers*, que não são imediatamente descodificados. Para além destes pacotes, cada nó mantém mais dois tipos de pacotes armazenados em *buffers*: pacotes que foram descodificados com sucesso e pacotes que necessitam de ser codificados e encaminhados.

A classificação de pacotes nos três tipos permite que o *DiffCode* explore o processo de descodificação no qual um nó codifica, através da operação *bitwise XOR*, um pacote codificado recebido com um pacote que não é imediatamente descodificado. Como tal, se a descodificação resultar num pacote nativo (não codificado), este pacote será movido para o *buffer* dos pacotes descodificados com sucesso.

O *DifCode* é capaz de obter uma taxa de redundância menor do que um protocolo de *Network Coding* probabilístico, em que os próximos nós de encaminhamento são escolhidos aleatoriamente.

3.3.2.2 Protocolos de *Single-hop Broadcast*

Ao contrário da transmissão em *Multi-hop*, na transmissão em *Single-hop*, quando um nó recebe um pacote, não retransmite imediatamente esse pacote. Em vez disso, o nó atualiza apenas as informações na sua base de dados, de acordo com o pacote recebido.

Para além disso, o nó transmite periodicamente apenas algumas informações da sua base de dados para os outros nós da rede. Assim, neste tipo de protocolos é necessário ter em conta o intervalo de transmissão e qual a informação necessária para transmitir, dando origem a duas categorias: abordagens de transmissão com intervalo fixo e abordagens de transmissão com intervalo adaptativo. Nas abordagens de intervalo fixo o principal desafio é a seleção e a agregação da informação importante a enviar, enquanto que nas abordagens de transmissão com intervalo adaptativo, para além deste desafio, existe um outro desafio, que diz respeito ao ajustamento dinâmico dos intervalos de envio da informação selecionada.

TrafficInfo e *TrafficView* são exemplos de protocolos baseados em intervalos fixos de transmissão *broadcast*. *Collision Ratio Controlo Protocol* (CRCP), *Abiding Geocast* e *Segment-oriented Data Abstration and Dissemination* (SODAC) são exemplos de protocolos baseados em intervalos adaptativos de transmissão *broadcast*. Estes protocolos não irão ser abordados, pois não serão o foco desta Dissertação.

Esta dissertação foca-se, sobretudo, em protocolos de disseminação em *Multi-hop broadcast*.

3.4 Considerações Finais

Neste capítulo foi feito um estudo sobre abordagens de distribuição de conteúdos, com vista a otimizar o tráfego de controlo. Foi dada uma atenção especial aos trabalhos que usaram *Bloom Filter*s, uma estrutura de dados que se mostrou útil em vários projetos, devido às suas capacidades de codificação e resumo. Em seguida, foi feito um estudo sobre possíveis estratégias para otimização do tráfego de dados em mecanismos de distribuição de conteúdos, cujo principal objetivo é a diminuição do problema de *broadcast storm*.

Capítulo 4

Algoritmos de Otimização de Conteúdos nas Redes Veiculares

4.1 Descrição do capítulo

No capítulo anterior fez-se o levantamento do estado de arte da literatura existente. Este capítulo fornece uma visão sobre o problema a ser resolvido, o qual deu origem a esta dissertação, e as várias soluções propostas, esclarecendo o conceito de cada solução, a sua implementação e aplicação. Serão estudadas as vantagens e desvantagens das estratégias atuais de disseminação já implementadas para compreender as possíveis melhorias que podem ser adicionadas ao mecanismo de disseminação de conteúdos, de forma a controlar a sobrecarga total na rede.

A organização deste capítulo é a seguinte.

- *secção 4.2*: *Problema* - descreve o problema que esta dissertação visa resolver, ou seja, conceber soluções para diminuição do tráfego de dados e controlo em mecanismos de disseminação de conteúdos em redes veiculares.

- *secção 4.3*: *Estratégias Atuais de Disseminação de Conteúdos* - descreve as estratégias atuais de disseminação já implementadas, destacando a estratégia com melhor taxa de entrega obtida.

- *secção 4.4*: *Soluções Propostas* - descreve as várias abordagens propostas para diminuir o tráfego de controlo, e as várias melhorias introduzidas ao nível dos pacotes de dados, resultando numa nova estratégia, de forma a diminuir o tráfego de dados.

- *secção 4.5*: *Considerações Finais* - apresenta o resumo deste capítulo.

4.2 Problema

Como referido anteriormente, as VANETs apresentam disrupções frequentes (ou conectividade intermitente), atrasos potencialmente longos (horas ou até mesmo dias), mudança rápida na topologia da rede, tempo de contacto entre os seus nós bastante reduzido, devido à constante mobilidade dos nós.

As DTNs constituem uma alternativa para lidar com as VANETs, através do mecanismo SCF. Este mecanismo pode ser usado pelos nós da rede, para que o conteúdo possa chegar ao seu destino, isto é, a todos os nós da rede. Os nós da rede armazenam os pacotes, carregando-os consigo e, quando a oportunidade apropriada surgir, os pacotes são transmitidos. Nas VANETs, a decisão de transmitir pacotes de dados pode basear-se em informações sobre os próprios nós vizinhos. Assim, surge o conceito de *Opportunistic Forwarding* [62], que consiste em decidir qual o pacote de dados a enviar a seguir, com base na informação previamente trocada entre os nós das proximidades, sobre o conteúdo armazenado em cada um deles.

Contudo, existem já algumas estratégias de disseminação implementadas, que seguem também o conceito de *Opportunistic Forwarding*, permitindo colocar alguma inteligência nos nós da redes, com o intuito de decidir quando enviar informação e qual informação a enviar. No entanto, este mecanismo ainda não está otimizado em relação aos nós que devem enviar informação, nem quanta informação pode estar duplicada na rede. Além disso, existem vários cenários possíveis: veículos parados nas estações (parques de estacionamento), em que tem de se limitar o número de mensagens e congestionamento na rede, e veículos na estrada, em que tem de se aumentar a quantidade de veículos a que se consegue chegar num determinado tempo.

Dentro deste contexto, esta Dissertação visa propor algoritmos de otimização para redução da sobrecarga da rede, aplicados às estratégias já implementadas. Como resultado, será proposta uma nova estratégia de distribuição de conteúdos com o objetivo principal de diminuir a sobrecarga da rede, tanto ao nível do tráfego como de dados e controlo, seguindo o conceito de *Opportunistic Forwarding*.

4.3 Estratégias Atuais de Distribuição de Conteúdos

Estão atualmente implementadas três estratégias de disseminação de conteúdos no grupo de investigação: *Least Number of Hops First*, (LNHF), *Local Rareste Bundle First* (LRBF) e *Local Rarest Generation First* (LRGF) [63].

A primeira estratégia mencionada, LNHF é baseada na seleção dos pacote de dados com base no seu número de saltos, para as OBUs e, com base no seu número de transmissões, para as RSUs. Assim, se o nó for uma OBU e se esta tiver vizinhos na sua zona de cobertura, a lista de pacotes a ser enviados é ordenada por ordem crescente com base no número de saltos dos pacotes, sendo que os pacotes com um número menor de saltos são enviados primeiro. Quando um vizinho recebe um pacote de dados atualiza as suas informações internas incrementando o número de saltos do pacote. Se o nó for uma RSU e se esta tiver OBUs na sua área de cobertura, a lista de pacotes a ser enviados é ordenada por ordem crescente com base no número de transmissões dos pacotes, sendo enviados em primeiro lugar os pacotes com um número menor de transmissões.

O principal propósito da estratégia LNHF é a associação do número de saltos de um pacote com o número de nós que já contém esse pacote: quanto maior o número de saltos de um pacote, maior a probabilidade de esse pacote específico estar armazenado nos outros nós.

Os resultados obtidos para a LNHF provaram que existe um baixo *delivery ratio*, quando comparada com as outras duas estratégias, porque não escolhe bem os pacotes a enviar, uma vez que não usa pacotes de controlo para saber quais os pacotes de dados a enviar a seguir.

Ao contrário da estratégia anterior, a estratégia LRBF foca-se em saber qual o conteúdo

armazenado em cada um dos seus vizinhos. Para isso, cada nó da rede envia pacotes de controlo para anunciar o conteúdo que tem armazenado, influenciando a decisão sobre quais pacotes de dados que devem ser enviados. A escolha dos pacotes de dados a enviar é feita pelos pacotes mais raros nos nós vizinhos.

Comparada com LNHF, esta estratégia consome mais recursos de rede devido à introdução dos pacotes de controlo. Contudo, os resultados mostram que a taxa de entrega é bastante mais elevada, pois cada nó tem informação dos pacotes que são necessários nos seus vizinhos.

O uso do *network coding* na estratégia LRGF traz um conjunto de vantagens, das quais se destacam: um melhor uso da largura de banda disponível, permitindo a maximização da capacidade de *multicast*, e a taxa de retransmissão é reduzida de $O(Nlog(N))paraO(N)$[64].

A estratégia LRGF tem um propósito semelhante à estratégia LRBF, uma vez que envia informação sobre a informação de que necessita. Usando o conceito de *network coding*, o ficheiro original é dividido em *frames* pela aplicação, que são reorganizados em blocos (identificados por *blockID*). Um bloco é um conjunto de *frames* adjacentes que são necessários para uma dada geração. Ao receber um pacote codificado, cada nó armazena o pacote na memória local para posterior descodificação. Para descodificar os todos os *frames* pertencentes ao mesmo bloco, um nó precisa de coletar um número superior ou igual de pacotes codificados, pertencentes à mesma geração, sendo os seus vetores de codificação linearmente independentes um do outro.

Cada nó anuncia periodicamente a geração e a percentagem de pacotes dessa geração que já tem, sendo esta informação usada na decisão de envio. Esta estratégia tem como objetivo diminuir a sobrecarga da rede, na medida em que os pacotes de controlo, embora estejam presentes, não anunciam todo o conteúdo de armazenamento de um nó, mas sim apenas a informação sobre a percentagem de pacotes que já tem de uma dada geração, ao contrário da estratégia LRBF: em vez de enviar um pacote específico para os vizinhos, tem de enviar um conjunto de pacotes codificados qualquer que sejam dessa geração.

Como resultado, a taxa de entrega é muito maior que na estratégia LNHF, embora seja ligeiramente menor quando comparada com a estratégia LRBF. Este facto deve-se a que na estratégia LRBF se saiba exatamente quais os pacotes que cada nó contém.

Devido a esta análise, esta dissertação apenas se vai focar na otimização da estratégia LRBF, uma vez que foi a estratégia que obteve uma maior taxa de entrega, mas que tem o problema de sobrecarga na rede devido aos pacotes de controlo. A próxima secção vai descrever detalhadamente esta estratégia, de forma e entender todo o seu funcionamento.

4.3.1 Local Rarest Bundle First

Como referido anteriormente, esta estratégia foca-se em saber qual o conteúdo armazenado em cada um dos seus vizinhos. Assim, cada nó da rede envia pacotes de controlo para anunciar o seu conteúdo armazenado, influenciando a decisão sobre quais os pacotes de dados que devem ser enviados.

Na Figura 4.1 é ilustrada a estratégia LRBF. Para tornar o exemplo mais simples, apenas é considerada a disseminação de um ficheiro, dividido em vários pacotes de dados. Assim, o nó S recebe os pacotes de controlo dos seus nós vizinhos, nós A e B, anunciando quais os pacotes de dados que já contêm armazenados até ao momento. O nó A anuncia no seu pacote de controlo a informação de que já tem os pacotes 1, 2 e 4. O nó B faz a mesma coisa que o nó A, anunciando que já tem os pacotes 1 e 4. Portanto, o nó S, fica a saber quais os pacotes de dados mais valiosos para a sua vizinhança. Dado que os pacotes 3 e 5 são os pacotes em falta,

e como o pacote 3 corresponde ao pacote mais raro, este é enviado em primeiro lugar, seguido dos pacote 5, 2 e 4. Tanto o envio de pacotes de controlo como de dados é feito em *Broadcast* para os seus vizinhos.

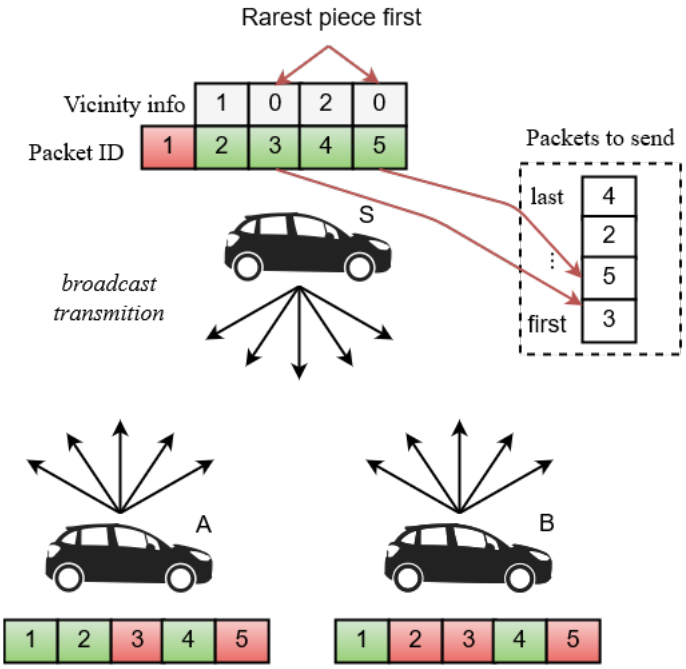


Figura 4.1: Estratégia LRBF, baseada em [12]

De forma a tornar todo o processo de decisão mais rápido quanto o possível, o nó S necessita de ter estruturas auxiliares internas para mapear os pacotes armazenados com o número de vizinhos que contêm um determinado pacote.

A Figura 4.2 mostra a estrutura dos pacotes de controlo, também designados por pacotes de anúncio (*advertisment*) usados pela LRBF. Cada campo corresponde a um tamanho de *bytes*. Dado que os pacotes de dados são identificados pelos seus *hashes*, a informação enviada no pacote de controlo são os *hashes* correspondentes aos pacotes de dados que o nó já contém. Estes *hashes* são equivalentes ao número de sequência do pacote.

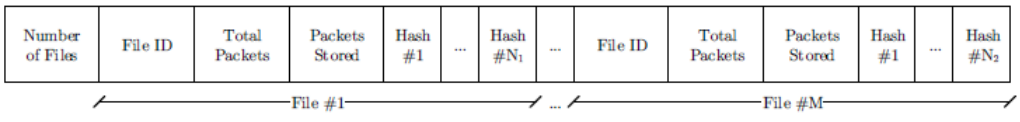


Figura 4.2: Estrutura do pacote de controlo de estratégia LRBF, baseada em [12]

O fator que determina o tamanho do pacote de controlo é o tamanho do ficheiro em pacotes a ser descarregado da rede. Isto significa que, se o nó tiver armazenado grande parte do ficheiro, o tamanho do seu pacote de controlo vai ser muito maior do que se tiver apenas armazenado uma pequena parte do ficheiro. Este aumento de tamanho introduz uma sobrecarga na rede considerável (*overhead*). Além disso, quanto maior a frequência com que os pacotes de controlo são enviados, maior é a sobrecarga da rede.

Por outro lado, a transmissão de pacotes de dados diminui ao longo do tempo, uma vez que os pacotes de controlo limitam a sua transmissão.

Contudo, quando os nós completam o ficheiro apenas enviam um pacote de controlo de pequeno tamanho, correspondendo apenas aos quatro primeiros campos da estrutura apresentada na Figura 4.2, em que os campos *TotalPackets* e *PacketStored* têm o mesmo valor.

A Figura 4.3 ilustra o procedimento de envio (a) e de receção (b) dos pacote de controlo.

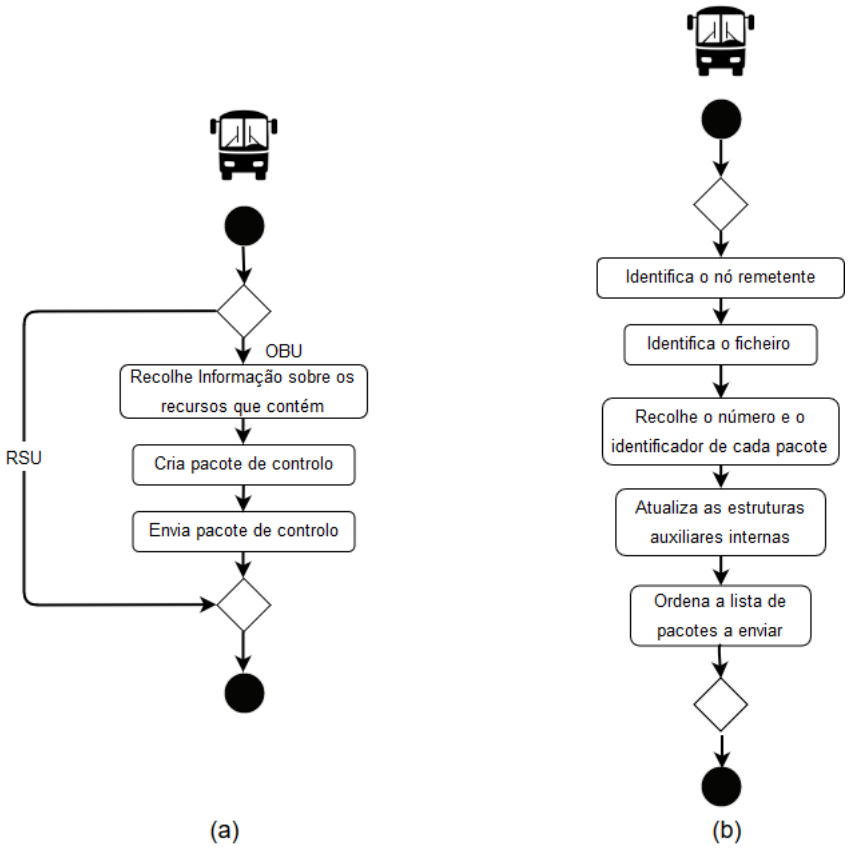


Figura 4.3: Procedimentos de envio (a) e receção dos pacotes de controlo da estratégia LRBF, baseada em [12]

A transmissão (a) e a receção (b) de pacotes de dados é apresentada na Figura 4.4. Na transmissão, o nó remetente depois de selecionar o pacote mais raro a ser enviado em primeiro lugar, incrementa o número de transmissões correspondente a esse pacote, enviando-o, logo de seguida. Posteriormente, ordena a lista de pacotes a serem enviados, para que o pacote enviado não seja novamente enviado.

Na receção de um pacote de dados, o nó verifica se já tem ou não o pacote. Em caso afirmativo, atualiza a entrada correspondente a esse pacote nas suas estruturas internas, ao que corresponde o incremento do número de vizinhos que contém este pacote. Caso contrário, cria uma nova entrada nas suas estruturas internas correspondente a esse pacote, armazenando-o de seguida. Em ambos os casos, é necessária a ordenação da lista de pacotes.

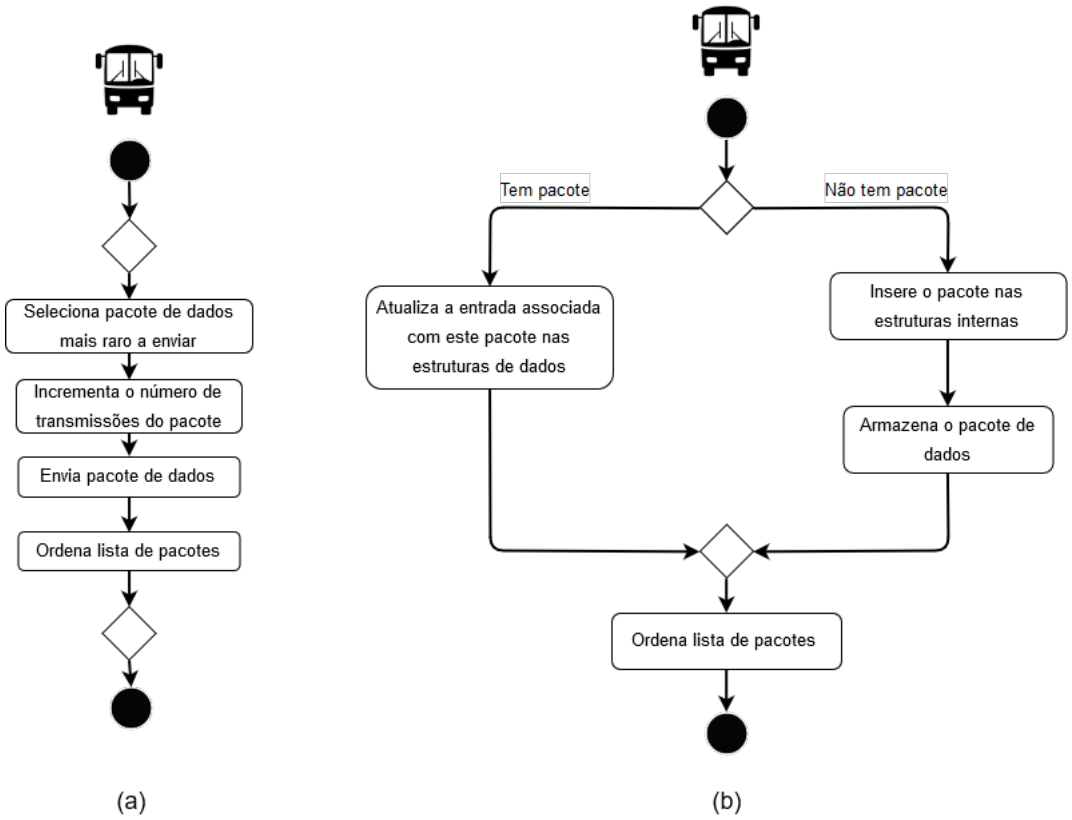


Figura 4.4: Procedimentos de envio (a) e receção dos pacotes (b) de dados da estratégia LRBF, baseada em [12]

4.4 Soluções Propostas

As soluções propostas podem ser aplicadas em qualquer estratégia que use pacotes de controlo para anunciar o conteúdo armazenado aos seus nós vizinhos. Para demonstrar a sua eficácia, estas soluções vão ser aplicadas na estratégia LRBF. Como referido anteriormente, esta estratégia foi a que se tornou mais eficiente. Em contrapartida, esta estratégia envia pacotes de controlo com um tamanho excessivamente grande, dependendo da quantidade de informação que o nó contém no seu armazenamento, levando a um aumento do *overhead* na rede.

Estas soluções visam resumir a quantidade de informação a ser enviada nos pacotes de controlo, diminuindo, consequentemente, o seu tamanho.

As subsecções seguintes vão descrever em pormenor cada solução proposta.

4.4.1 Otimização através do Algoritmo de Bandas

Muitas vezes a informação em falta nos nós da rede corresponde a conjuntos de pacotes consecutivos. O algoritmo de Bandas considera que em vez de enviar todos os pacotes em falta, envia apenas o primeiro e último pacote dos conjuntos de pacotes consecutivos, ou seja, envia a informação em Bandas.

Considerando que os pacotes de dados são identificados pelos seus *hashes*, pode-se consi-

derar i e f, como sendo o início e fim de cada banda, respetivamente. Para um conjunto de pacotes armazenados por uma OBU, $S = \{p_1, p_2, ..., p_n\}$, sendo n o número total de pacotes desse conjunto, representando a sequência de pacotes ordenados por ordem crescente (não sendo necessariamente o número total de pacotes que contém cada ficheiro), esta abordagem considera que a informação necessária para ser enviada no pacote de controlo é apenas o par formado por $\{p_1, p_n\}$, em vez de todo o conjunto S contendo todos os *hashes* dos pacotes. Sendo assim, $i = p_1$ e $f = p_n$, formando uma banda (B) (ou um par).

Se existirem mais conjuntos $S_1 = \{p_1,p_2,...,p_n\}$, $S_2 = \{p_1,p_2,...,p_n\}$, ..., $S_m = \{p_1,p_2,...,p_n\}$, sendo m o número total de conjuntos, a sua aplicação vai ser $i_0 = p_1S_1$, $f_0 = p_nS_1$, formando $B_1 = \{i_0, f_0\}$, $i_1 = p_1S_2$, $f_1 = p_nS_2$, formando $B_2 = \{i_1, f_1\}$, ..., $i_{m-1} = p_1S_m$, $f_{m-1} = p_nS_m$, formando $B_m = \{i_{m-1}, f_{m-1}\}$. Nestes casos onde o conjunto S é composto por apenas um só elemento, $S = \{p_1\}$, ambos i_{m-1} e f_{m-1} contêm p_1, formando $B_m = \{p_1, p_1\}$. A entrega desordenada de pacotes de dados é comum nas *VANET*s porque a grande mobilidade de nós gera conexões intermitentes. Contudo, o contacto com os vários vizinhos na sua área de cobertura, vai fazer com que os nós preencham as "falhas"na sequência ordenada de *hashes* dos pacotes, até finalmente obterem a totalidade dos pacotes que um determinado ficheiro contém. Ao preencheram estas "falhas", o número de bandas, m vai diminuindo, formando bandas maiores.

A Figure 4.5 apresenta a estrutura do pacote de controlo, denominada *bands vector*, resultante desta abordagem *Bandas*.

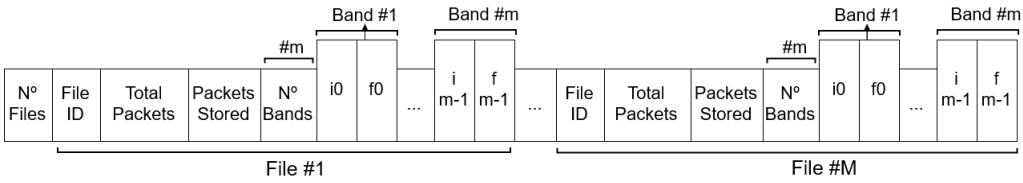


Figura 4.5: Estrutura do pacote de controlo otimizado com *Bandas*

Como se pode verificar, em vez de ser enviada a informação de todos os pacotes presentes, apenas é enviada a informação do primeiro e último pacote de cada conjunto de pacotes consecutivos, o que diminui o *overhead* principalmente quando muitos dos pacotes já foram enviados.

4.4.2 Otimização através do Algoritmo de *Bit Array*

O segundo algoritmo proposto codifica a informação sobre os *hashes* dos pacotes armazenados por cada nó, usando um array de bits, onde cada bit indica se um determinado pacote já foi ou não recebido. O tamanho do array de bits (número de bits) corresponde ao tamanho do ficheiro dividido em pacotes, conferindo assim um tamanho fixo. Se existir mais do que um ficheiro para disseminar na rede, vai haver mais do que um array de bits para representar a informação de cada ficheiro. Por exemplo, considerando a disseminação de um ficheiro de 75MB dividido em 2256 pacotes, o tamanho o array de bits vai ser de 2256. Este array pode ser representado por um *int (32-bit) array [71]*, considerando mais 16 bytes para ser múltiplo de 4 bytes.

A Figura 4.6 seguinte mostra que cada elemento do array tem 4 blocos. Este array pode ser usado para informar sobre 2272 pacotes de dados. Este tamanho é o tamanho normal para

o conteúdo a ser transferido nas redes veiculares, como vídeos promocionais ou atualizações de software.

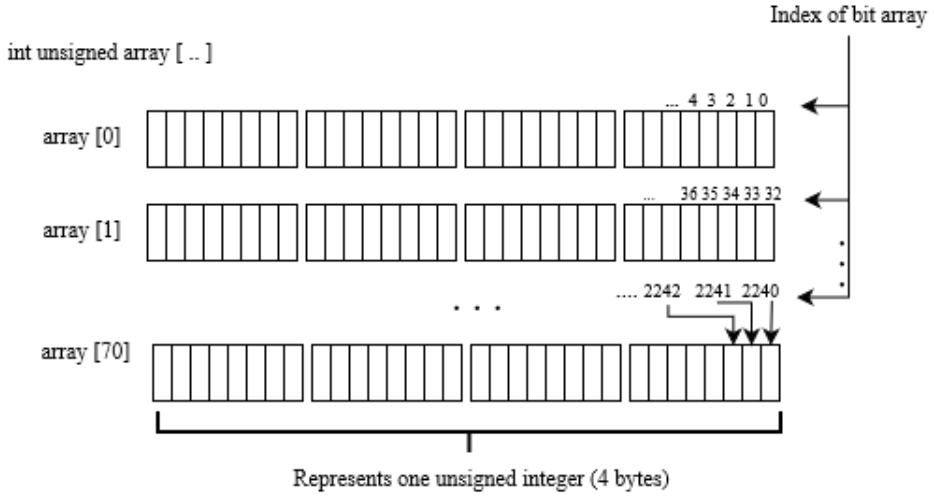


Figura 4.6: Estrutura do *Bit Array*

A Figura 4.7 apresenta a estrutura do pacote de controlo, designada *bitCode vector*, resultante da abordagem *bit array*.

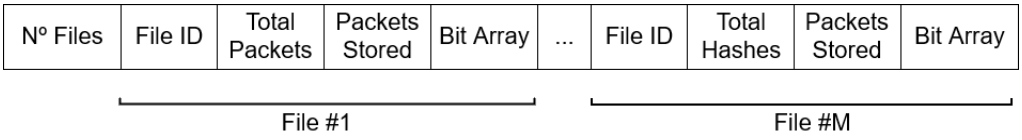


Figura 4.7: Estrutura do pacote de controlo otimizado com *Bit Array*

O tamanho fixo do pacote de controlo proposto por esta abordagem permite a aplicação de outras otimizações, como funções de compressão, para minimizar ainda mais a informação, o que será considerado como trabalho futuro.

4.4.3 Otimização através de *Bloom Filter*

Os *Bloom Filters* oferecem uma forma sucinta de representar um conjunto ou uma lista de itens. Um *Bloom Filter* oferece uma representação que pode reduzir significativamente o espaço com o custo da introdução de falsos positivos. Se os falsos positivos não causarem problemas significativos, o *Bloom Filter* poderá fornecer um bom desempenho. Além disso, os *Bloom Filters* oferecem vantagens em termos de espaço quando comparados com outras estruturas de dados, por exemplo *hash tables*, uma vez que não armazenam os pacotes reais, bem como a vantagem de tempo, devido à sua consulta rápida.

Dado um conjunto $S = \{x_1,x_2,...,x_n\}$ de n elementos, um *Bloom Filter* consiste na representação do conjunto S através de um array de m bits, inicialmente inicializados todos a '0'. Um *Bloom Filter* usa k funções de hash independentes $h_j(x)$: $N \rightarrow \{1,2,...,m\}$, em que $j \in \{1,2,..,k\}$. Estas funções de hash mapeiam cada elemento do conjunto S para um número aleatório uniforme no intervalo {1,2,...,m}, que corresponderá a uma posição no array de bits.

Para cada elemento do conjunto S, $x \in S$, os bits $h_j(x)$ são colocados a '1'. Uma posição pode ser definida como '1' várias vezes, mas somente a primeira alteração tem efeito.

Para verificar se um elemento y pertence ao conjunto original S, todas as posições indicadas por cada $h_j(y)$ são avaliadas com o objetivo de verificar se foram ou não ativadas. Se existir pelo menos uma destas posições $h_j(y)$ em que não foi colocada a '1', então conclui-se que o elemento y claramente não pertence ao conjunto S. Se todas as $h_j(y)$, estão a '1', conclui-se que o elemento y pertence ao conjunto S com uma certa probabilidade de falsos positivos. Portanto, um *Bloom Filter* pode gerar um falso positivo, onde sugere que um elemento x está em S, embora não seja. A Figura 4.8 ilustra um exemplo de um *Bloom Filter*.

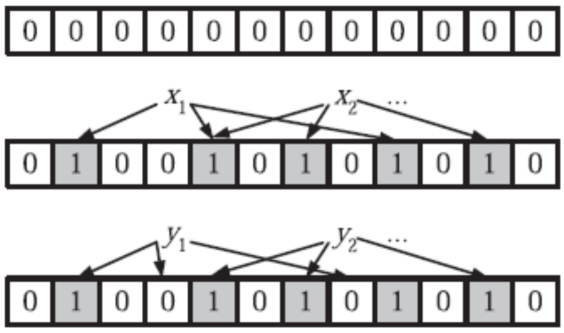


Figura 4.8: Um exemplo de *Bloom Filter*, segundo [13]. O filtro começa como um array de zeros. Cada elemento do conjunto S, x_i é dividido k vezes, em que cada *hash* produz uma posição de bits. Estes bits são definidos a '1'. Para verificar se um elemento y está no conjunto, é necessário o *hash* k vezes e verificar os bits correspondentes. Assim, o elemento y_1 não pode estar no conjunto, pois um dos bits define um '0'. O elemento y_2 está no conjunto ou o filtro gerou um falso positivo.

Para muitas aplicações, os falsos positivos podem ser aceitáveis desde que a sua probabilidade seja suficientemente pequena. Assumindo que as funções de *hash* são aleatórias, a taxa de falsos positivos (f_p) para um array de tamanho m, com número de elementos n e número de funções *hash* k, de um *Bloom Filter* é definida pela equação:

$$f_p = [1 - (1 - \frac{1}{m})^{kn}]^k \approx [1 - e^{-\frac{kn}{m}}]^k = (1 - f_p)^k \tag{4.1}$$

A economia de espaço e comunicação oferecida pelo *Bloom Filter* muitas vezes supera a sua desvantagem de permitir falsos positivos.

A Figura 4.9 mostra a estrutura do pacote de controlo, designada *bloomCode vector*, resultante desta abordagem.

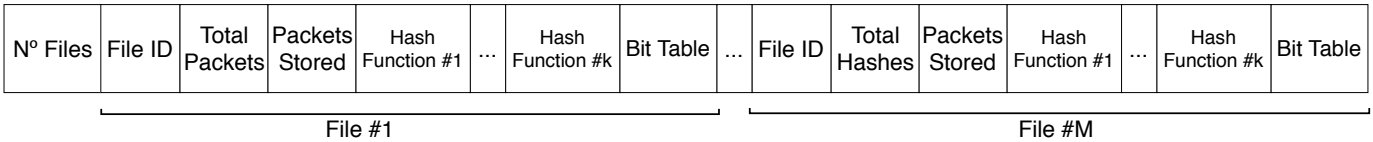


Figura 4.9: Estrutura do pacote de controlo otimizado com *Bloom Filter*

Em suma, o *Bloom Filter* é semelhante a uma *hash table*, com a exceção de que não armazena os valores reais, mas coloca os respetivos *bits* a '1' no array de *bits*, previamente

inicializado a zeros.

Usando o *bloom filter* a única informação a enviar no pacote de controlo será o array de *bits* com os *hashes* dos pacotes que o nó contém, juntamente com o número de funções *hash*. O cabeçalho do pacote de controlo irá incluir as funções *hash* usadas para mapear cada *hash* do pacote no array e o número máximo de elementos que pode ser inserido no filtro. O nó emissor cria o *bloom filter* estabelecendo o número máximo de pacotes a serem inseridos no filtro, calcula o tamanho apropriado do array de bits, o número de funções *hash* e a probabilidade de falsos positivos. Depois insere os *hashes* dos pacotes que já tem no filtro. Como resultado, o pacote de anúncios é construído e enviado em broadcast para seus vizinhos. Com esta informação o nó recetor do pacote será capaz de replicar o *bloom filter* do nó emissor e assim atualizar e ordenar a lista de pacotes de dados a serem enviados consoante esta informação.

4.4.4 Estratégias para diminuição do número de pacotes de dados

Quando os nós da rede enviam pacotes de dados em *broadcast*, ocorre o problema de *broadcast storm*, levando ao congestionamento da rede. Este problema surge na estratégia LRBF, pelo envio de grande quantidade de pacotes de dados redundantes.

Nesta subsecção é feita uma análise onde posteriormente são apresentadas várias formas de melhoria da estratégia LRBF para ultrapassar este desafio. Como resultado, surge uma nova estratégia chamada *Local Rareste Bundle First Advanced (LRBFAdvanced)* baseada na *LRBF*, com a integração da abordagem que se tornou mais eficiente em reduzir o tamanho dos pacotes de controlo, bem como as várias otimizações efetuadas ao nível da diminuição do congestionamento introduzido pelo envio dos pacotes de dados.

Em [12] são propostas duas formas de minimizar o problema de *broadcast storm* para a estratégia LRBF. A primeira é baseada no número de saltos que um pacote teve, como forma de estimar a presença de um pacote na rede. Assim, quanto maior for o número de saltos que um pacote teve, maior a probabilidade de este pacote já estar armazenado numa grande quantidade de nós. Por outro lado, se o número de saltos de um pacote for baixo, significa que este pacote deve ser enviado de modo a aumentar a sua presença na rede. A segunda forma baseia-se no tipo de nó que o nó transmissor tem na sua vizinhança. Se o nó transmissor for uma OBU e tiver uma RSU como vizinha, não envia pacotes. Isto deve-se à grande probabilidade de os nós vizinhos da OBU terem também a RSU como vizinha, recebendo assim, o mesmo conteúdo sob disseminação.

Contudo estes dois mecanismos não são suficientes. Ora vejamos o porquê: voltando à Figura 4.4 (a), um nó, quer sendo uma OBU (quando não tem na sua vizinhança RSUs) quer uma RSU, quando envia um pacote apenas incrementa o número de transmissões, ordenando em seguida a lista de pacotes a ser enviada, pelo pacote mais raro, para uma determinada vizinhança. O pacote não é marcado como enviado, podendo ser reenviado novamente, o que leva ao aumento de redundância de pacotes de dados enviados.

A primeira otimização passa por um nó marcar para cada pacote enviado, que esse mesmo pacote foi realmente enviado, mantendo todo o processo existente. Caso esse pacote não tenha sido recebido pelos nós vizinhos, o nó transmissor é informado através do pacote de controlo, desmarcando esse pacote como enviado para o poder tornar a enviar novamente (Figura 4.10 (a)). Este processo de desmarcação é feito sempre apenas para todos os pacotes que não foram recebidos. Esta desmarcação também é feita sempre que a vizinhança mude, ou seja, sempre que um nó vizinho deixe de ser vizinho do nó transmissor por um período de tempo

bem definido, para impedir que o pacote nunca mais seja transmitido (Figura 4.10 (b)).

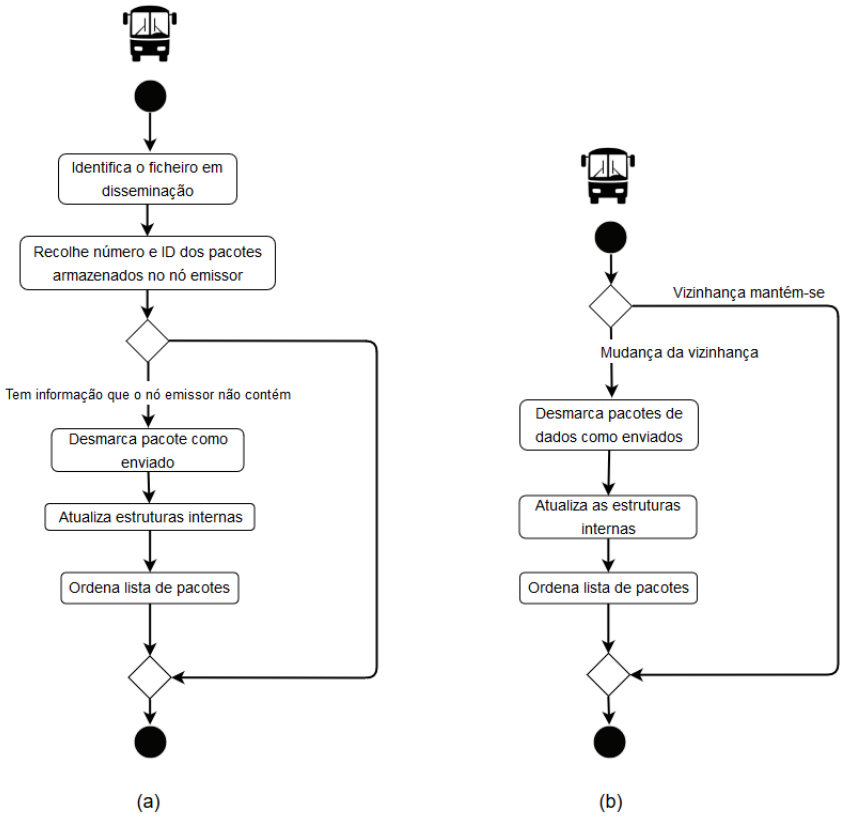


Figura 4.10: Procedimentos de desmarcação de um pacote de dados como enviado aquando da receção de um pacote de controlo (a) e na mudança da vizinhança de um nó (b) na estratégia LRBFAdvanced.

Sempre que um nó recebe e/ou "ouve"um pacote de dados na rede, marca esse pacote como enviado, independentemente de ter ou não esse pacote armazenado. Assim, já não será retransmitido novamente. A Figura 4.11 ilustra o procedimento de marcação de um pacote de dados sempre que a vizinhança mude.

Voltando novamente à Figura 4.1, embora seja feita a ordenação da lista de pacotes a serem enviados, por pacotes mais raros, quando todos os pacotes mais raros forem enviados, o que vai acontecer é que vão ser enviados os pacotes que toda a vizinhança já tem, isto é, o pacote 2 e 4 vão ser enviados, mesmo estando posicionados no fundo da lista. Isto acontece porque o objetivo da estratégia *LRBF* é ordenar apenas por pacote mais raro e não impede que os pacotes de dados já armazenados pela vizinhança não sejam enviados, resultando no envio de pacotes redundantes, ocupando a largura de banda desnecessariamente. Como resultado, o pacote de dados apenas será enviado se a estrutura que informa quantos vizinhos tem determinado pacote for inferior ao número de vizinhos, mantendo assim, o envio dos pacotes mais raros em primeiro lugar.

Como tal, foi necessário alterar o procedimento de envio e receção de pacotes de dados. A Figura 4.12 mostra todas as alterações efetuadas.

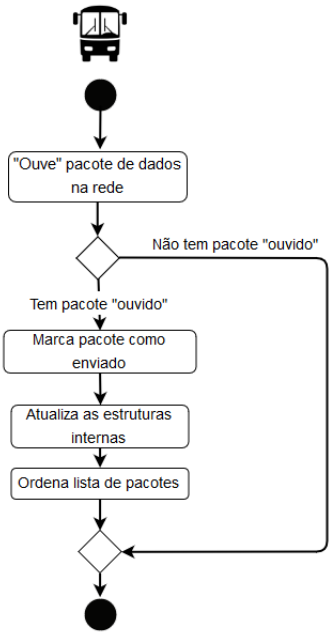


Figura 4.11: Procedimento de marcação de um pacote como enviado, sempre que um nó "ouve"um pacote, na estratégia LRBFAdvanced.

4.5 Considerações Finais

Neste capítulo foram descritas as estratégias atuais de disseminação já implementadas, destacando a estratégia com melhor taxa de entrega obtida. Seguidamente, foram descritas várias soluções para diminuir o tráfego de controlo nos mecanismos de distribuição de conteúdos. Por fim, são identificados vários problemas, na estratégia com melhor desempenho, que levam ao aumento da redundância de pacotes de dados na rede. Com isto, são propostas um conjunto de otimizações ao nível da diminuição do tráfego de dados, para diminuir a sobrecarga na rede. Como resultado das otimizações introduzidas é proposto um nova estratégia, LRBFAdvanced.

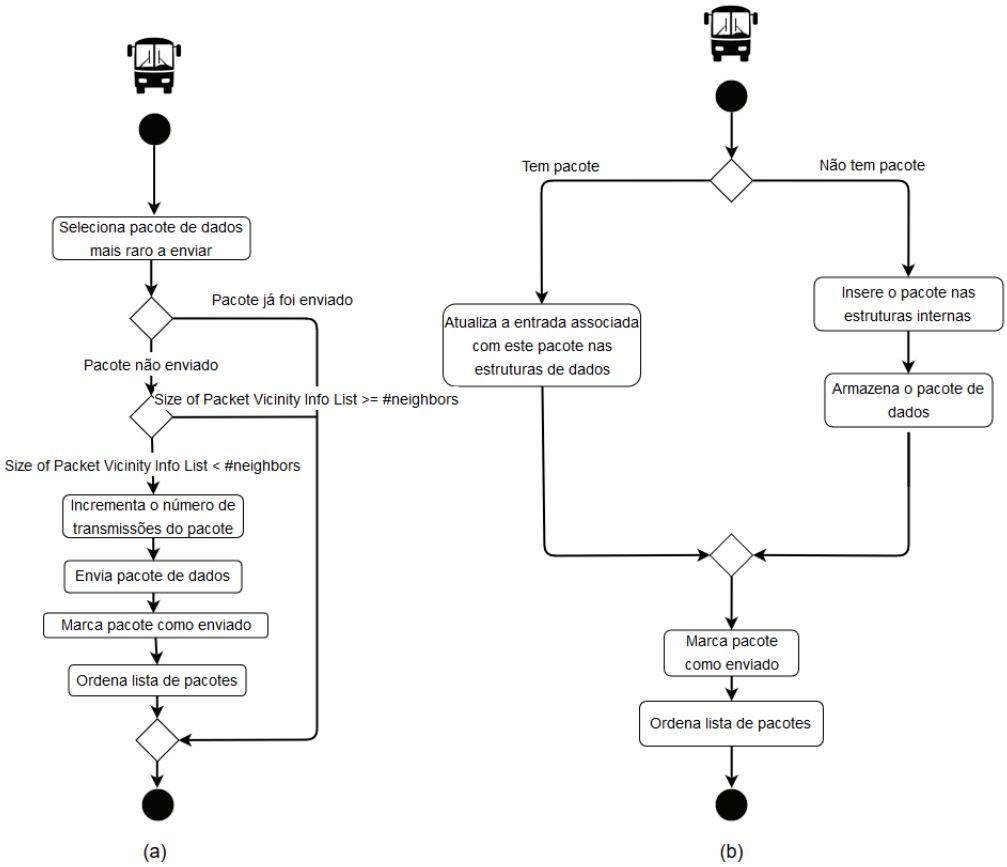


Figura 4.12: Procedimentos de envio (a) e receção dos pacotes (b) de dados da estratégia LRBFAdvanced

Capítulo 5

Implementação e Integração

5.1 Descrição do capítulo

Neste capítulo será apresentada a arquitetura e o funcionamento do mOVERS utilizado para avaliar as soluções propostas para diminuição do tráfego de dados e controlo. Serão descritos os módulos existentes no mOVERS, bem como as suas funções.

De seguida, serão apresentadas todas as modificações efetuadas de forma a ser possível implementar as quatro soluções propostas para diminuição do tráfego de controlo, assim como a nova estratégia resultante das otimizações para diminuição do tráfego de dados. Será também, apresentada a forma como foi feita a integração do código fonte do mOVERS com as placas reais, OBUs e RSUs. Sendo assim, este capítulo está organizado da seguinte forma.

- *secção 5.2*: *Descrição do Emulador mOVERS* - descreve os módulos do emulador bem como as suas funções.
- *secção 5.3*: *Modificações Globais* - descreve todas as modificações efetuadas ao emulador, descreve também a implementação de todas as soluções propostas, bem como os seus algoritmos.
- *secção 5.4*: *Implementação do algoritmo de otimização através de Bandas* - descreve a implementação da primeira solução proposta, designada *Bandas*, bem como os seus algoritmos.
- *secção 5.5*: *Implementação do algoritmo de otimização através de BitArray* - descreve a implementação da segunda solução proposta, designada *BitArray*, bem como os seus algoritmos.
- *secção 5.6*: *Implementação do algoritmo de otimização através de Bloom Filters* - descreve a implementação das soluções propostas que usam *Bloom Filters*, designadas *BFfixedSize* e *BFvariableSize*, bem como os seus algoritmos e as principais diferenças entre elas.
- *secção 5.7*: *Implementação da estratégia LRBFAdvanced* - descreve a implementação da estratégia resultante, bem como as principais modificações introduzidas.
- *secção 5.8*: *Integração do código fonte nas placas NetRider* - descreve a integração do código fonte do emulador nas placas *NetRider*.

- *secção 5.9* : *Considerações Finais* - apresenta o resumo deste capítulo.

5.2 Descrição do Emulador mOVERS

O emulador mOVERS foi concebido através de uma parceria entre o grupo de investigação NAP do IT e a Veniam® para desenvolver uma nova implementação do protocolo *bundle*. Esta implementação foi concebida para operar nas redes veiculares, particularmente, na rede do Porto, usando por base uma DTN.

O software do mOVERS foi desenvolvido na linguagem de programação C/C++ e suporta comunicações usando protocolos WAVE (IEEE 802.11p), entre veículos; Suporta também comunicações usando tecnologias Wi-Fi (IEEE 802.11a/b/g).

Todos os algoritmos de otimização desenvolvidos foram implementados e avaliados nesta plataforma, pois o código desenvolvido neste emulador tem a particularidade de ser igualmente aplicável tanto nos veículos emulados (OBUs e RSUs), como a nível das placas das OBUs e RSUs reis.

A arquitetura do mOVERS está representada na Figura 5.1 e foi concebida para ser modular e extensível. É composta por sete módulos tais como *Neighboring, Socket, API Management, Storage, RX* e *Routing*, em que cada um deles é responsável por executar as funções necessárias de forma a operar como uma DTN. Para executar tais funções, os módulos têm de interagir entre si através dos *Inter-Process Communication IPC Sockets*. O protocolo UDP é usado para que um nó possa trocar pacotes com outros nós da rede.

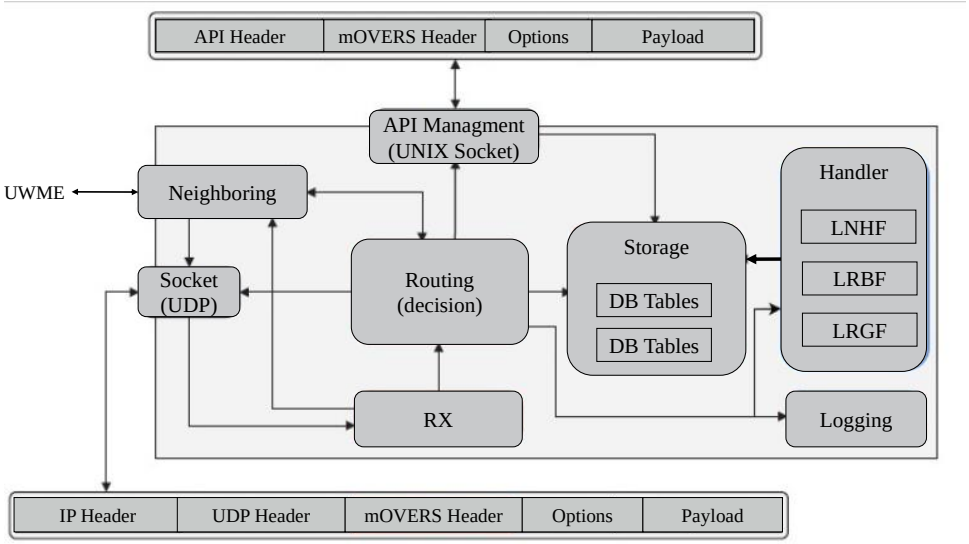


Figura 5.1: Arquitetura do mOVERS [12]

O mOVERS permite a criação de vários processos que executam o software DTN, em que cada um deles emula um único nó da rede, um veículo.

Para que o mOVERS se tornasse escalável foi introduzida a framework de filas de mensagens ZMQ [65]. Assim, é possível lidar com um número elevado de processos, em que cada processo corresponde a um nó da rede.

Existem diversos tipos de mensagens que são trocados entre o emulador e os nós da rede, tais como: mensagens de controlo, enviadas pelo emulador ou pacotes regulares (pacotes de dados, pacotes de controlo, relacionadas com informação de dados e de vizinhança), trocads entre os nós da rede.

Quando um nó da rede recebe uma mensagem de controlo, as informações como a posição do nó e a sua lista de vizinhos são atualizadas. No entanto, o módulo *Socket* não recebe a mensagem de controlo, uma vez que não corresponde a um pacote regular. Sempre que um pacote regular for recebido, este é encaminhado para o módulo *Socket*, sendo posteriormente classificado pelo módulo *RX*. Consoante esta classificação, o pacote será depois encaminhado para o módulo *Routing* ou *Neighboring*.

Assim, para que a troca de mensagens entre os processos seja possível, foi criada a classe **EmuMessageHeader**, como mostra a Figura 5.2.

Quando um nó envia informação para outros nós da rede, é criado um objeto desta classe em que os seus campos são inicializados. Este objeto corresponde ao cabeçalho junto com o pacote que vai ser enviado, sendo utilizado pelo emulador para encaminhar o pacote para o seu destino.

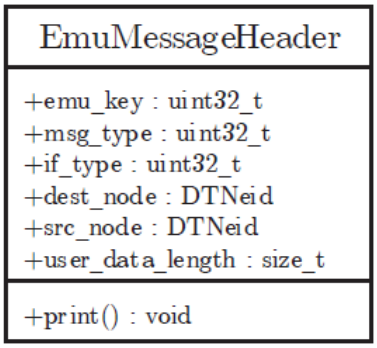


Figura 5.2: Classe mOVERS [12]

Seguidamente irão ser descritos cada um destes módulos, nas subsecções seguintes.

5.2.1 Módulos de Comunicação

Os módulos *RX* e *Socket* são considerados como módulos de comunicação, uma vez que lidam com a entrada e saída de pacotes de controlo ou de dados. É importante referir que podem existir dois tipos de pacotes de controlo, nomeadamente, pacotes de controlo relativamente ao controlo da vizinhança e pacotes de controlo relativamente ao controlo de dados, mas ambos pertencentes ao tipo de pacote regular.

O módulo *Socket* fornece uma camada de abstração para o envio ou receção de pacotes de ou para nós vizinhos e gere o acesso ao *Socket* UDP.

O módulo *RX* tem internamente uma *thread* responsável por verificar constantemente se foi recebido algum pacote através do *Socket* UDP. Em caso afirmativo, este módulo analisa e classifica os pacotes de acordo com a sua *flag* (**DTN_FLAG_NEIGH_ACK_MASK, DTN_FLAG_END_ACK_MASK** e **DTN_FLAG_ADV_MASK**), o destino (**dstEID**) e o ID do serviço (controlo de vizinhança ou distribuição de conteúdo). Feita esta classificação, o módulo *RX* encaminha os pacotes para o módulo *Routing*, caso os pacotes tenham sido

classificados como pacotes de dados, ou de controlo de dados, ou para o módulo *Neighboring*, caso os pacotes tenham sido classificados como controlo de vizinhança.

5.2.2 Módulo *Neighboring*

O módulo *Neighboring* é responsável pela descoberta dos nós vizinhos. Pode operar com diferentes tipos de vizinhos dependendo da interface de comunicação que está a ser usada. A comunicação pode ser feita através das tecnologias Wi-Fi, WAVE e interfaces Ethernet. Tendo sido desenvolvido para comunicações veiculares, os vizinhos WAVE são tipicamente RSUs ou OBUs, os nós com interfaces Wi-Fi dizem respeito a sensores ou terminais e as RSUs e servidores, situados no núcleo da rede, estão conectados através de interfaces Ethernet.

Um novo tipo de vizinho é também definido como *Static*, sendo responsável por lidar com as rotas pré-definidas entre os nós da rede.

A inicialização deste módulo é feita pelo módulo *Routing*. Cada nó envia periodicamente pacotes de controlo de vizinhança, tendo como objetivo anunciar a sua presença na rede. Desta forma, todos os nós que recebem um pacote de controlo de vizinhança atualizam as suas tabelas internas com as informações dos seus vizinhos em cada instante de tempo (*Endpoint Identifier*(EDI), endereço IP, tipo - RSUs ou OBUs, porto de comunicação e o valor *Received Signal Strength Indicator (RSSI)*).

5.2.3 Módulo *Storage*

O módulo *Storage* é responsável por armazenar vários tipos de pacotes que são importantes para a decisão de encaminhamento da informação. Este módulo está no centro de toda a atividade que acontece em torno de um nó. Pode ser consultado pelo módulo *Routing* para averiguar a existência de um determinado pacote e pode receber um novo pacote.

Este módulo permite que, ao mesmo tempo, um nó envie e receba um pacote de dados de/para a sua *Storage*, aquando da execução do algoritmo de encaminhamento de pacotes de dados pelo módulo *Routing*. Todo este processo acontece de forma concorrente, assim como alguns módulos que executam as suas próprias *threads*, não havendo sincronismo entre eles. Este módulo tem se ser capaz de lidar com todos estes processos. Para isso, cada operação pública é designada de *thread-safe*, o que quer dizer que as *threads* podem bloquear enquanto outra operação está a ser executada em simultâneo.

A Figura 5.3 mostra como este módulo está organizado, podendo-se observar dois tipos de armazenamento: *StorageDisk* e *StorageRAM*. É possível observar, ainda, uma organização lógica que consiste em dois tipos: *StorageData* e *StorageInfoTables*.

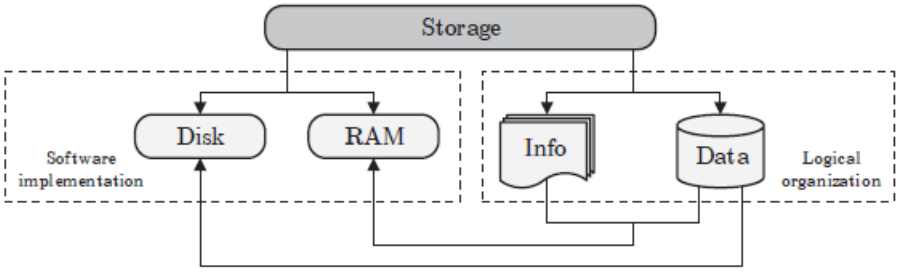


Figura 5.3: Organização do módulo *Storage* [12]

O *StorageDisk* diz respeito ao armazenamento persistente, mantendo os pacotes de dados e as informações adicionais necessárias para os manipular e encaminhar. Implementa uma tabela designada de *hashTable*, que organiza os pacotes armazenados por identificador do pacote.

O *StorageRAM* implementa quatro tipos diferentes de tabelas em memória, designadas de *StorageInfoTables* com o objetivo de fornecer um acesso rápido e uma pesquisa otimizada dos pacotes de dados. Estas tabelas são:

- *Expiry Table*: onde os pacotes estão organizados por ordem crescente de tempo de expiração;
- *NoData Table*: diz respeito aos pacotes que são conhecidos na rede, mas não existem dados guardados sobre eles.
- *OnHold Table*: onde os pacotes estão organizados por ordem crescente de tempo de espera até que sejam enviados. Estão também organizados por ordem crescente de tempo de expiração.
- *Own Table*: onde os pacotes estão organizados pelo tipo de serviço, designado *serviceID* em que o destino é o próprio nó.

O *StorageData* é responsável pela gestão do armazenamento persistente dos pacotes no disco, lê e escreve ficheiros, um por cada pacote, para extrair o conteúdo do pacote. Usa as tabelas implementadas no *StorageRAM* para melhorar o desempenho dessas operações.

5.2.4 Módulo *Routing*

O módulo *Routing* é responsável por implementar as decisões sobre quais os pacotes que devem ser enviados, para quais vizinhos, e em que momento. Assim, este módulo implementa a lógica de envio e receção dos pacotes de cada estratégia implementada.

O mOVERS adotou uma solução híbrida de encaminhamento, uma vez que este módulo encaminha os pacotes consoante o tipo de vizinhos, RSU ou OBU, e o tipo de pacote, controlo ou dados.

A primeira decisão a ser tomada é baseada no tipo de pacote. No entanto, o restante processo depende do tipo de nó, pelo que existem quatro classes implementadas que são apresentadas na Figura 5.4. Sempre que um nó quer enviar um pacote, verifica se existem vizinhos disponíveis; se sim, seleciona um ou mais pacotes de dados da sua *Storage*, isto é, do módulo *Storage*.

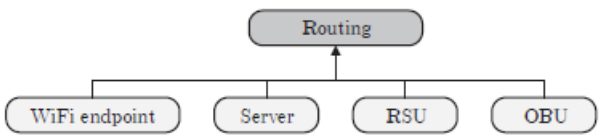


Figura 5.4: Classes implementadas consoante o tipo de encaminhamento [12]

5.2.5 Módulo API *Management*

O módulo API *Management* é responsável por tornar possível a interação entre os nós do mOVERS e as aplicações externas. Para isso, este módulo usa mensagens IPC para separar mensagens de dados de mensagens de controlo.

Este módulo tem uma *thread* que é responsável por lidar com os pacotes das aplicações do mOVERS através da API *Socket*; é também responsável por criar uma camada de abstração para enviar/receber pacotes do mOVERS para a/da API. Além do mais, gere o acesso das aplicações ao mOVERS.

5.2.6 Módulo *Logging*

O módulo de *Logging* é responsável por efetuar a recolha de todos os dados e informações que são precisas para avaliar o desempenho das estratégias de disseminação de conteúdos implementadas. Assim, as variáveis usadas para armazenar essas informações são apresentada em baixo.

- **log_id**: identificador único correspondente ao ficheiro de *log*.
- **node_eid**: identificador EDI do nó onde a informação está a ser gerada.
- **timestamp**: instante de tempo em formato *UNIX* pelo qual a informação foi gravada.
- **packets_stored_total**: número total de ficheiros armazenados pelo nó desde o início do processo de disseminação.
- **packets_transmitted_total**: número total de pacotes transmitidos pelo nó desde o início do processo de disseminação.
- **packets_stored_per_timestamp**: número de pacotes armazenados durante o instante temporal referido.
- **packets_transmitted_per_timestamp**: número de pacotes transmitidos no instante temporal referido.
- **packets_listened_per_timestamp**: número de pacotes "ouvidos"no meio *wireless* durante o instante temporal referido.
- **packets_recv_good_per_timestamp**: número de novos pacotes recebidos e armazenados pelo nó.
- **packets_recv_bad_per_timestamp**: número de pacotes recebidos pelo nó dos quais já tinha armazenados.
- **packets_rejected_good_per_timestamp**: número de pacotes que o nó não tinha armazenados mas que foram rejeitados devido ao limite da largura de banda.
- **packets_rejected_bad_per_timestamp**: número de pacotes que o nó já tinha armazenados mas que foram rejeitados devido ao limite da largura de banda.
- **control_packets_number_per_timestamp**: número total de pacotes de controlo por instante temporal referido.

- **control_packets_size_per_timestamp**: tamanho total dos pacotes de controlo transmitidos por instante temporal referido.

Este módulo é crucial para a avaliação do desempenho das otimizações de disseminação de conteúdos implementadas, pois permite o cálculo das métricas de avaliação do desempenho.

5.2.7 Módulo *Handler*

O módulo *Handler* é responsável por conter todas as classes auxiliares, com as estruturas de dados internas necessárias para a implementação de cada uma das estratégias de disseminação de conteúdo. Estas classes são, posteriormente, chamadas no módulo *Routing*, consoante a estratégia que se queira emular.

Neste momento, estão implementadas as estratégias *Least Number of Hops First*, (LNHF), *Local Rareste Bundle First* (LRBF) e *Local Rarest Generation First* (LRGF). Como referido anteriormente, apenas as duas últimas estratégias mencionadas, LRBF, foram alvo de estudo.

5.2.8 UMWE

Quando o mOVERS é executado nas placas NetRider, para a rede veicular real, estas placas usam uma aplicação designada *Universal WAVE Management Entity* (UMWE) [66], que por sua vez, usa uma abordagem multi-canal para transmissão. Como referido anteriormente, dois tipos de canais são especificados pelo WAVE: o canal de controlo (CCH) e o canal de serviço (SCH). O CCH opera continuamente, e quando o mOVERS necessita de realizar uma tarefa de comunicação, é enviada uma solicitação para aceder ao canal SCH.

5.2.9 Recolha de dados da Base de Dados MySQL

Para que os nós da rede possam atualizar o seu estado com nova informação, o emulador envia periodicamente mensagens de controlo para todos os nós. Esta informação corresponde à informação real recolhida da rede veicular do Porto e está armazenada numa base de dados *MySQL*, sendo constantemente consultada.

As Figuras 5.5, 5.6 e 5.7 mostram como esta informação está organizada na base de dados, pelo que houve a necessidade de criar três Tabelas: (1) Tabela que contém informação organizada por *timestamp*, respeitante a cada nó; (2) Tabela que está relacionada com a tabela anterior, na medida em que o nó referenciado na Tabela anterior contém, para o *timestamp* referenciado, informação dos seus vizinhos, identificador e valor do RSSI do(s) vizinho(s); (3) Tabela referente à informação que cada RSU tem, para os instantes em que os dados foram recolhidos da rede veicular do Porto.

Entry ID	UNIX Timestamp	Node ID	GPS coordinates	Direction	Speed	Cell	Cell RSSI

Figura 5.5: Tabela *timestamps* da Base de Dados do mOVERS [12]

Entry ID	UNIX Timestamp	Neigh ID	Link RSSI

Figura 5.6: Tabela *neighbors* da Base de Dados do mOVERS [12]

RSU ID	location	Helix version	Board version	owner	vnetwork	provider	uptime	lastping	GPS coordinates

Figura 5.7: Tabela *rsu* da Base de Dados do mOVERS [12]

O mOVERS recupera a informação da base de dados em formato JSON com a ajuda do servidor Apache e do *Hypertext Preprocessor* PHP.

Existem duas APIs importantes disponíveis, a primeira tem o objetivo de obter a lista de RSUs, e a segunda tem o objetivo de obter o estado atual da rede para cada instante de tempo. Para isso, basta fazer dois pedidos HTTP GET à base de dados. Posteriormente, o mOVERS analisa a informação em formato JSON e define uma mensagem de controlo para enviar a todos os nós para que possam atualizar o seu estado, cuja classe está representada na Figura 5.8. Esta informação é extremamente importante, uma vez que contém as posições de cada um dos nós, a sua lista de vizinhos, entre outros.

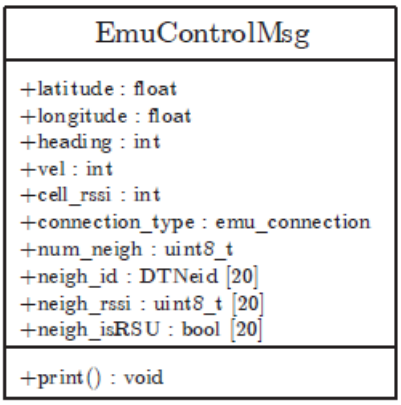


Figura 5.8: Classe que representa as mensagens de controlo enviadas pelo mOVERS a todos os nós da rede [12]

Os dados reais recolhidos da rede veicular implementada no Porto, designados de *datasets*, devem ser importados para a base de dados, para que possam ser usados pelo mOVERS, através de um script. No entanto, antes de importar estes dados, é necessário confirmar as especificações do script.

5.2.10 Estrutura dos Pacotes

A estrutura dos pacotes mOVERS é a seguinte:

- Cabeçalho mOVERS
  - Versão: versão do emulador;
  - ServiceID: identifica o serviço ao qual o pacote pertence: *Neighbour Discovery or Content Distribution service*;
  - Remetente e Destinatário EDIs: identificadores do nó remetente e do nó destinatário do pacote;
  - Informação de destino: tipo de nó (por exemplo);

  - EDI Anterior: identificador do nó que era detentor do pacote;
  - Hash: identificador único do pacote;
  - Data de Expiração: tempo de criação e tempo de vida do pacote;
  - Tamanho dos Dados: tamanho do pacote de dados;
  - Tamanho do Payload: tamanho do pacote de controlo;
  - Prioridade;
  - Número de vizinhos: número de vizinhos que contém o pacote;
  - Flags: identificador do tipo de pacote;
  - ID do Ficheiro: identificador único do ficheiro;
  - Número Total de Pacotes do Ficheiro: número total de pacotes que o ficheiro, identificado no campo anterior, ocupa;
  - Identificador do Bloco: apenas é necessário na estratégia LRGF;
  - Tamanho do Bloco: referente ao número de pacotes que codifica um bloco. Apenas é necessário na estratégia LRGF;
  - Tamanho da Geração: referente ao número de pacotes que codifica uma geração. Apenas é necessário na estratégia LRGF;

- Payload: array de bytes com tamanho máximo de 32KB.

5.2.11 Modo de Operação

O mOVERS é baseado num software *multi-thread* composto por diversos módulos que operam de forma concorrente.

Como é apresentado na Figura 5.9, o primeiro passo é a configuração do mOVERS. Esta configuração é efetuada através de *scripts*, onde são definidos os parâmetros necessários para executar o mOVERS. Estes parâmetros são a porta do módulo *Socket*, capacidade do módulo *Storage* e seu caminho, ficheiros de *log* e seus caminhos, interfaces de comunicação, a lista de nós da rede a ser usada, quer de OBUs quer de RSUs, a versão de encaminhamento que se quer testar, assim como o dataset a ser usado (ver secção). Toda esta informação vai ser usada para a criação e inicialização de todos os módulos do mOVERS. Aquando da criação de cada módulo é criada e executada uma nova *thread* com o objetivo de desempenhar o comportamento pelo qual o módulo foi construído.

Como já foi referido anteriormente, a partir do momento em que cada *thread* é lançada, ela compete de forma concorrente pelos recursos da máquina onde está a ser executada.

A experiência está em execução até ser enviado um sinal externo, tipicamente um comando para parar toda a experiência. Quando o emulador recebe este comando, pára a execução de todas as *threads*, guardando o contexto da experiência.

5.3 Modificações Globais

De modo a implementar as soluções propostas para diminuir o *overhead* introduzido pelos pacotes de controlo e de dados, várias modificações tiveram de ser feitas à arquitetura do mOVERS. A Figura 5.10 mostra as alterações efetuadas à arquitetura inicial do mOVERS (ver Figura 5.1). As alterações foram feitas nos módulos *Routing, mOVERS Packet Header e*

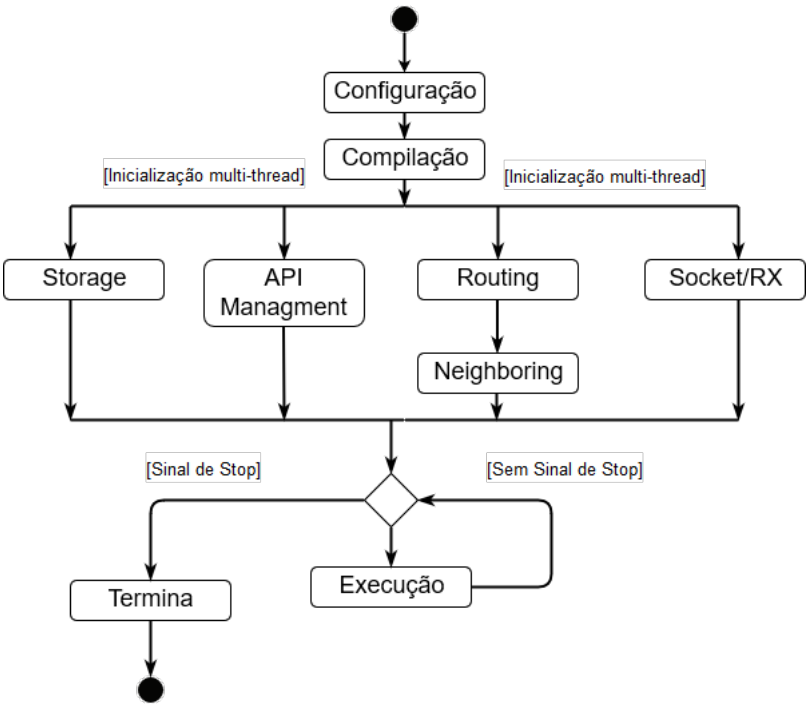


Figura 5.9: Fluxograma de modo de operação do mOVERS baseado em [12]

Handler. No módulo *Handler* foram introduzidos cinco módulos auxiliares: Bands (LRBF + otimização com Bandas), BitArray (LRBF + otimização com Bit Array), *BFfixedSize* (LRBF + *Bloom Filter* com tamanho fixo), *BFvariableSize* (LRBF + *Bloom Filter Variable Size* com tamanho variado) e por fim, *LRBFAdvanced* (contendo a melhor abordagem de otimização do tráfego de controlo (*BitArray*) + LRBF com as várias otimizações ao nível do tráfego de dados). Além disso, um novo tipo de pacote de controlo é introduzido em *Bands, BitArray* (Figuras 4.5 e 4.7) e nos *Bloom Filters* (Figura 4.9).

5.3.1 Biblioteca de suporte ao mOVERS

O emulador tem uma biblioteca de suporte, chamada *libmOVERS*, que define várias funções, assim como algumas especificações gerais a serem usadas pelas aplicações. O propósito desta biblioteca é definir a estrutura do cabeçalho de pacote do mOVERS. Para ser possível a integração das abordagens propostas, algumas modificações ao cabeçalho do pacote tiveram de ser feitas, adicionando novos campos:

- **randomSeed**: representa um valor que assegura o mesmo comportamento para cada função de hash;
- **numberOfHashFunctions**: representa o número de funções de hash necessárias para mapear cada elemento no filtro, ou seja, o valor de k;
- **numberOfElementsProjected**: representa o número de elementos que foram inseridos no filtro.

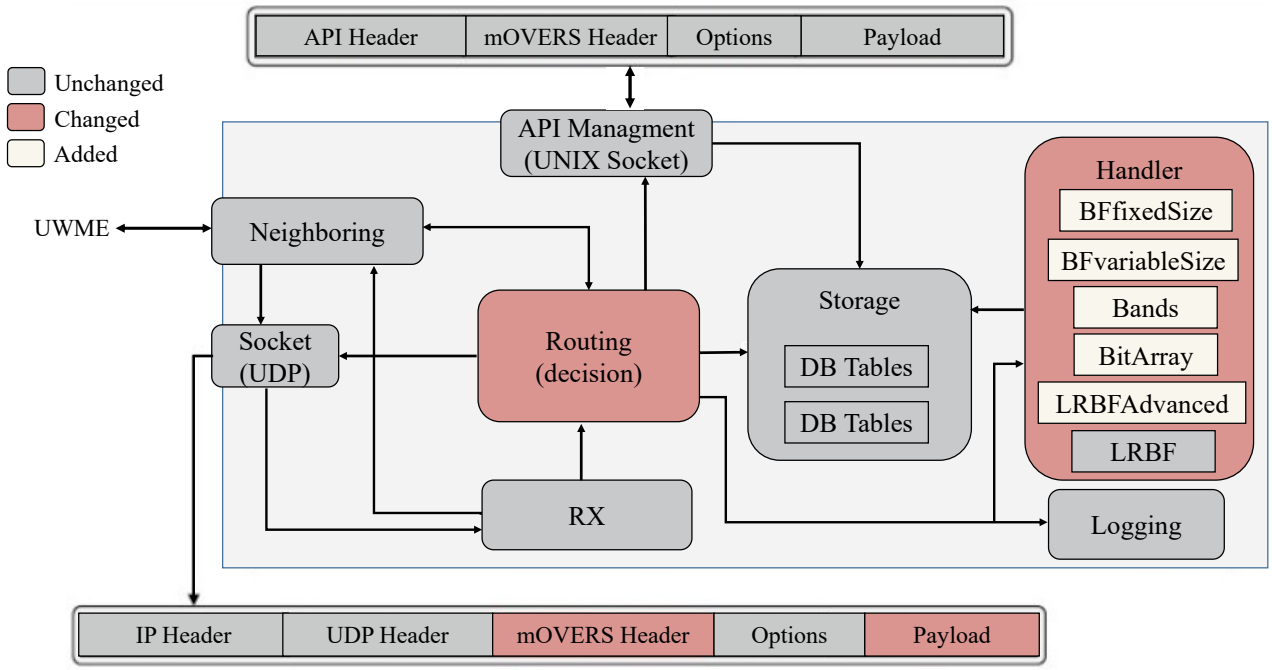


Figura 5.10: Arquitetura do mOVERS com as abordagens propostas integradas, baseada em [12]

- **FalsePositiveProbability**: representa o valor da probabilidade de falsos positivos que o filtro contém;

5.3.2 Módulo de Limite de Largura de Banda

O módulo de limite de largura de Banda para o emulador, responsável por impor um máximo de receção e transmissão de pacotes por cada nó, apenas incluía os pacotes de dados. Desta forma, foi necessária a sua modificação para ser extensível, também, para os pacotes de controlo. Assim, sempre que um nó recebe um pacote de dados ou de controlo verifica primeiro se tem largura de banda disponível para o receber. Se dispuser de largura de banda, o nó processa o pacote; caso contrário, o pacote é descartado. A mesma analogia é feita para o caso da transmissão de pacotes de dados ou de controlo. Primeiro verifica se tem largura de banda disponível. Se sim, então o pacote é transmitido; caso contrário, não transmite o pacote.

Desta forma, o mOVERS está preparado para executar as simulações de modo a comparar os resultados da estratégia original LRBF com as soluções propostas nas mesmas condições.

As soluções propostas implementadas no mOVERS consistem em novas classes envolvidas no processo de transmissão e receção de pacotes de controlo, sem qualquer alteração no processo de transmissão e receção dos pacotes de dados. Na secção seguinte são apresentados os passos necessários para a implementação das quatro estratégias propostas no emulador mOVERS.

5.4 Implementação do algoritmo de otimização através de *Bandas*

Numa vizinhança os nós trocam pacotes de controlo para que todos saibam quais os pacotes de dados que já estão armazenados em cada um deles. Ao longo do processo de disseminação, a estratégia LRBF usa um campo do pacote de controlo para sinalizar cada pacote de dados já armazenado (Figura 4.2). No entanto, à medida que a disseminação progride, é possível identificar intervalos sequenciais de pacotes de dados já recebidos. A estratégia *Bands* usa essa característica, considerando apenas o início e o final de cada sequência (Figura 4.5). Para implementar esta abordagem, é necessário alterar o processo de preenchimento do pacote de controlo (*Algoritmo de Encaminhamento*) para o envio das informações, e o processo de análise do pacote de controlo recebido (*Algoritmo de Receção*).

Conforme descrito no Algoritmo 1, a transmissão de um pacote de controlo ocorre quando o nó tem a sua *flag canSendControlPackets* ativa, e disponibilidade de largura de banda para transmissão (1). O número de ficheiros em disseminação é inserido no vetor de bandas (2). Para cada um dos ficheiros (3), o *FileID* (4), o número total de pacotes que o ficheiro ocupa (5) e o número de pacotes que o nó já contém armazenados (6) são inseridos no primeiro campo do vetor de bandas (Figura 4.5 – FILE #M).

Se o nó não recebeu a totalidade dos pacotes de dados correspondentes a um determinado ficheiro (7), ordena o seu armazenamento por ordem crescente (8), iterando sobre ele de seguida (9). Para cada pacote do seu armazenamento obtém o seu identificador (*hash*). Na formação da primeira banda (11) insere o primeiro pacote num array temporário (*tmp_array*) (12). Depois, percorre todos os pacotes do seu armazenamento até encontrar uma falha na sequência (15). O último pacote antes da falha da sequência é considerado o fim da banda atual (16); o primeiro pacote depois da falha é considerado o começo da próxima banda (17). Caso não haja falhas na sequência crescente, é formada apenas uma banda, sendo o último pacote do seu armazenamento o fim da banda (19). Este processo é repetido para todos os pacotes armazenados.

Finalmente, o número total de bandas (*Bands*) formadas é inserido no vetor bandas (20). Por sua vez, cada banda formada é inserida a seguir (21). Assim, o pacote de controlo é criado e preenchido com o vetor bandas (23).

O processo de receção começa após a receção de um pacote de controlo (Algoritmo 2). O nó recetor obtém o número de ficheiros em disseminação (2), e para cada um deles (3), obtém o número total de pacotes do ficheiro (5) e o número de pacotes já armazenados pelo nó vizinho (6). Se o ficheiro estiver incompleto (7), isto é, o campo *Packet Stored* é menor do que o campo is *Total Packets* (Figura 4.5), obtém o número total de bandas formadas pelo nó vizinho (8), iterando, de seguida através delas (9). Para cada banda obtém i_{m-1} (10) e f_{m-1} (11), correspondendo ao início e fim. Para cada pacote correspondente a cada banda, obtém o seu *hash* (13), e verifica se está presente ou não no seu armazenamento e estruturas de dados (15). Caso o nó recetor já contenha esse pacote, as informações relativas à sua vizinhança são atualizadas (15), e a sua lista de pacotes de dados a serem enviados a seguir é ordenada consoante essas informações (16). Depois, o nó recetor verifica se contém informações que o nó transmissor não contenha (17). Em caso afirmativo, o nó recetor ativa a sua *flag canSendDataPackets* (18) para o envio dos novos pacotes de dados.

Caso o ficheiro esteja completo (23), o nó recetor verifica apenas se o ficheiro é conhecido ou não (24). Caso não seja, insere informações do novo ficheiro no seu armazenamento (24)

Algorithm 1 Control Packet Forwarding Band

```
1:  if canSendControlPackets and !bandwidthFull then
2:      Get and insert total number of filesID known in bands vector
3:      for file = firstFile to file = lastFile do
4:          Insert fileID in bands vector
5:          Insert total packets in bands vector
6:          Insert packets stored in bands vector
7:          if file is not completed then
8:              Sort packet storage
9:              for packet=firstPacket to packet=lastPacket do
10:                 Get packet hash
11:                 if packetHash == firstPacketHash then
12:                     Insert packetHash in tmp_array
13:                 if packetHash != lastPacketHash then
14:                     fecth and get nextPacketHash
15:                     if compare(nextPacketHash - packetHash) > 1 then
16:                         Insert packetHash in tmp_array
17:                         Insert nextPacketHash in tmp_array
18:                 else
19:                     Insert packetHash in tmp_array
20:             Insert total numbers of bands in bands vector
21:             Insert all elements of tmp_array in bands vector
22:     if bands vector was filled then
23:         Create and fill control packet with bands vector
24:         Send control packet in broardcast
25:         Logging
```

e ativa a *flag canSendControlPackets*, para o envio de pacotes de controlo.

Algorithm 2 Control Packet Reception Band

```
1:  if received Packet then
2:      Peek number of Files
3:      for file = firstFile to file = lastFile do
4:          Peek fileID
5:          Peek total packets
6:          Peek packets stored
7:          if File is not completed then
8:              Peek neighbour's total number of bands
9:              for band = firstBand to band = totalNumberOfBands do
10:                 Peek neighbour's packet initial band
11:                 Peek neighbour's packet final band
12:                 for packet = packetInicialBand to packet = packetFinalBand do
13:                     Get packet hash
14:                     if packet with this hash already in structure and in storage then
15:                         Update vicinity info
16:                         Sort list of data packets to send to neighbours
17:                     else
18:                         if there is a new file to donwload then
19:                             Set canSendControlPacket flag to true
20:             if receiver has different hashes to send then
21:                 Set canSendDataPackets flag to true
22:         else
23:             if File doesn't exist in storage then
24:                 Add file's info to storage
25:                 Set canSendControlPackets flag to true
```

5.5 Implementação do algoritmo de otimização através de *BitArray*

A abordagem *BitArray* consiste num *array* de *bits*, e a informação de cada pacote é codificada em apenas um *bit*, que indica se o pacote já foi ou não recebido (Figure 4.7). Para implementar esta abordagem é necessário alterar o processo de preenchimento do pacote de controlo para envio da informação (*Algoritmo de Encaminhamento*) e o processo de análise do pacote de controlo recebido (*Algoritmo de Receção*).

De forma similar à abordagem anterior, a transmissão de um pacote de controlo ocorre quando o nó tem ativa a sua *flag canSendControlPackets* e tem largura de banda disponível para a transmissão (1), conforme é descrito no Algoritmo 3. Além disso, é de salientar que, em ambas as abordagens, um pacote de controlo só é transmitido se existirem nós vizinhos na sua área de cobertura. O número de ficheiros em disseminação é inserido no vetor *bitCode* (2). Para cada ficheiro (3), o *FileID* (4), o número total de pacotes do ficheiro (5), e o número de pacotes já armazenados pelo nó são inseridos no vetor *bitCode* (Figure 4.7 – FILE #i).

Se o ficheiro não estiver completo (7), é calculado o tamanho do array de bits (8), tendo em conta o número total de pacotes que o ficheiro ocupa. Antes do processo de preenchimento, o array de bits é inicializado com zeros (9). De seguida, para cada pacote (10) já armazenado no nó, obtêm-se o seu *hash* (11), ativando o respetivo bit no array de bits (12).Este processo é repetido até que não haja mais pacotes armazenados. Finalmente, o *array* de *bits*, depois de preenchido, é inserido no vetor *bitCode* (14).

Algorithm 3 Control Packet Forwarding Bit Array

```
 1: if canSendControlPackets and !bandwidthFull then
 2:     Get and insert total number of filesID known in bitCode vector
 3:     for file = firstFile to file = lastFile do
 4:         Insert file ID in bitCode vector
 5:         Insert total packets in bitCode vector
 6:         Insert packets stored in bitCode vector
 7:         if file is not completed then
 8:             Calculate bit array size
 9:             Fill bit array with zeros
10:             for packet=firstPacket to packet=lastPacket do
11:                 Get packet hash
12:                 bitarray[(hash/32)]| = (1 << (hash%32))
13:     if bit array was filled then
14:         Insert bit array in bitCode vector
15:         Create and fill control packet with bitCode vector
16:         Send control packet in broardcast
17:         Logging
```

O processo de receção começa depois da receção de um pacote de controlo (Algoritmo 4). O nó recetor obtém o número de ficheiros que o nó transmissor contém (2). Então, para cada um dos ficheiros (3), obtém o *fileID* (4), o número total de pacotes que o ficheiro ocupa (5) e os pacotes já armazenados (6) pelo nó transmissor.

Se o ficheiro não estiver completo (7) obtém o *array* de *bits* do nó transmissor (8). Para cada posição no *array* de *bits* (9), obtém o valor de cada bit (10)Se o valor da posição do bit for 1 (11), obtém o *hash* de cada pacote (12). Na verdade, a posição de cada bit é iniciada em 1, uma vez que não existem *hashes* de pacotes com o número zero. De seguida, verifica se contém o pacote no seu armazenamento e nas suas estruturas (13). Se sim, a informação relativa aos vizinhos é atualizada (14) e a lista de pacotes de dados a enviar a seguir é ordenada consoante

essa informação (15). Caso contrário, verifica se o pacote pertence a um novo ficheiro (17), ativando a *flag canSendControlPackets* (18), em caso afirmativo. Seguidamente, verifica se o nó transmissor contém toda a informação que o nó recetor contém (19). Se não contiver toda a informação, a *flag canSendDataPackets* é ativada para o envio de pacotes de dados mais raros (20).

Se o ficheiro está completo (21), o nó recetor apenas verifica se conhece ou não o ficheiro (22). Se não conhecer o ficheiro insere as informações relativas ao ficheiro no seu armazenamento (23), ativando a *flag canSendControlPackets* (24). Assim termina o processo de receção de um pacote de controlo.

Algorithm 4 Control Packet Reception Bit Array

```
 1: if received Packet then
 2:     Peek number of Files
 3:     for file = firstFile to file = lastFile do
 4:         Peek fileID
 5:         Peek total packets
 6:         Peek packets stored
 7:         if File is not completed then
 8:             Peek neighbour's bit array
 9:             for k_bit = firstK_bit to k_bit = lastK_bit do
10:                 bitarray[(k_bit/32)]&(1 << (k_bit%32)))! = 0
11:                 if bit is set then
12:                     Get packet hash with k_bit
13:                     if packet with this hash already in structure and in storage then
14:                         Update vicinity info
15:                         Sort list of data packets to send to neighbours
16:                     else
17:                         if there is a new file to donwload then
18:                             Set canSendControlPacket flag to true
19:             if receiver has different hashes to send then
20:                 Set canSendDataPacket flag to true
21:         else
22:             if File doesn't exist in storage then
23:                 Add file's info to storage
24:                 Set canSendControlPacket flag to true
```

5.6 Implementação dos algoritmos de otimização através de *Bloom Filters*

De forma similar às outras estratégias, para implementar esta abordagem foi necessário modificar o processo de criação do pacote de controlo para o envio da informação (algoritmo de encaminhamento) e o processo de análise do pacote de controlo recebido (algoritmo de receção).

Como *Border and Mitzenmacher* referiram em [13], há muitos lugares na rede onde se pode manter ou enviar uma lista, mas a lista completa exigiria muito espaço. A estratégia LRBF é um exemplo do envio da lista completa de pacotes.

O *bloom filter* usado é uma classe que contém os seguintes atributos: *saltCount, tableSize, projectedElementCount, insertedElementCount, randomSeed, bitTable* e *desiredFalsePositiveProbability* sugerida por *Arash Partowin* [67]. *saltCount* diz respeito ao número de funções *hash* necessárias para a criação do filtro; *tableSize* corresponde ao tamanho da *bit table* (m) do filtro; *projectedElementCount* diz respeito ao número máximo de elementos (pacotes) que se espera inserir no filtro; *insertedElementCount* corresponde ao número real de elementos

que foram de facto inseridos no filtro; *randomSeed* é necessária para adicionar aleatoriedade às funções de *hash*, de forma a não terem o mesmo comportamento, mapeando cada bit numa posição diferente no filtro; *bitTable* corresponde ao *bloom filter* em si, contendo toda a informação codificada; e por último *desiredFalsePositiveProbability* diz respeito à probabilidade de falsos positivos desejada. Todos estes membros são inicializados a zero e preenchidos quando o filtro é criado. Os métodos principais envolvidos na criação do *bloom filter*, preenchendo estes atributos, são *computeOptimalParameters*, *effectiveFPP, insert, contains, hashing* e *saltGenerator*.

Da mesma forma que as abordagens anteriores, a transmissão de um pacote de controlo acontece sempre que um nó tenha a sua *flag canSendControlPackets* (1) ativa, como é descrito no Algoritmo 5. O número total de ficheiros conhecidos pelo nó é inserido no vetor *bloomCode* (Figure 4.9 – FILE #i). Para cada um desses ficheiros (3), os campos ID do ficheiro (4), o número total de pacotes (5) e o número de pacotes já armazenados (6) são preenchidos. Se o ficheiro não estiver completo (7), os parâmetros do *bloom filter* são definidos (8), consoante o tipo de *bloom filter*. Sempre que um novo filtro é criado, alguns parâmetros têm de ser definidos à *priori*. Esses parâmetros são *projectedElementCount, desiredFalsePositiveProbability* e *randomSeed*. O *projectedClementCount* é predefinido consoante o tipo de *bloom filter*, de tamanho fixo ou tamanho variável. Para o *bloom filter* de tamanho fixo este parâmetro é fixo e é predefinido como o número total de pacotes em que o ficheiro é dividido. Pelo contrário, para *bloom filter* de tamanho variável, este parâmetro é variável, pois é predefinido como o número de pacotes que um nó tem armazenados num determinado momento, sendo igual ao número real de pacotes inseridos no filtro (*insertedElementCount*). A *desiredFalsePositiveProbability* também tem de ser predefinida, de forma a não gerar falsos positivos, bem como a *randomSeed*.

O número mínimo ótimo de funções hash (*saltCount*) e o tamanho mínimo da *bitTable* necessário, tendo em conta o número de elementos a serem inseridos, são calculados através do método `computeOptimalParameters`. Uma vez calculado o número de funções hash necessárias para o filtro, um número correspondente de *randomSeed* é definido para ser usado.

Os métodos *hashing* e *saltGenerator* correspondem às operações necessárias para transformar o identificador de um pacote (*hash*) num *bit* a ser inserido numa determinada posição na *bitTable*. O método *saltGenerator* é responsável por gerar diferentes *randomSeeds* a partir de um conjunto predeterminado, de modo que uma única função de *hash* (presente no método de *hashing*) possa gerar instâncias de *bloom filters* exclusivas ao inserir um elemento na *bitTable*. A função de *hash* usada foi a *AP Hash Function* sugerida por *Arash Partowin*, em que aplica um *hybrid rotative and additive hashing algorithm*.

Prosseguindo no Algoritmo 5, por cada pacote armazenado, o seu *hash* é inserido no filtro (12). O método *insert* funciona de forma similar, enviando um *hash* do pacote, através da função de *hashing*, várias vezes (indicado pelo *saltGenerator*) e preenchendo os respetivos bits da *bitTable*.

Assim, uma vez preenchido o *bloom filter* (13), isto é, se o nó tem pacotes armazenados de qualquer ficheiro, as funções de *hash* usadas e a *bitTable* são inseridas no vetor *bloomCode* (14).

Como tal, o pacote de controlo é criado e preenchido com o vetor *bloomCode* (15). Os parâmetros do *bloom filter*, como o número de funções de *hash*, *projectElementCount* e a *randomSeed* são inseridos no cabeçalho do pacote (16). O nó transmissor tem de assegurar que quando um nó vizinho recebe o seu pacote de controlo, tem de ser capaz de replicar o *bloom filter* com a informação inserida no pacote de controlo.

Depois disso, o pacote de controlo está pronto para ser enviado em *broadcast* (17).

Algorithm 5 Control Packet Forwarding Bloom Filter

```
1:  if canSendControlPackets and !bandwidthFull then
2:      Get and insert total number of files known in bloomCode vector
3:      for file = firstFile to file = lastFile do
4:          Insert file ID in bloomCode vector
5:          Insert total number of packets in bloomCode vector
6:          Insert number of packets stored in bloomCode vector
7:          if file is not completed then
8:              Set bloom filter parameters
9:              Create bloom filter with parameters
10:             for packet = firstPacket to packet = lastPacket do
11:                 Get packet hash
12:                 Insert packet hash in bloom filter
13:     if bloom filter was filled then
14:         Insert hash functions and bit table in bloomCode vector
15:         Create and fill control packet with bloomCode vector
16:         Insert bloom filter parameters in packet's header
17:         Send control packet in broardcast
18:         Logging
```

Da mesma forma que as abordagens anteriores, o processo de receção acontece sempre que é recebido um pacote de controlo. O nó recetor começa por obter o número de ficheiros que o nó transmissor conhece (2). Depois, por cada ficheiro conhecido (3), obtém o *fileID* (4), o número total de pacotes que o ficheiro ocupa (5) e os pacotes armazenados (6).

Se o ficheiro não esta completo (7), o nó recetor recolhe os parâmetros do cabeçalho do pacote (8). Como referido anteriormente, o nó recetor tem de ser capaz de replicar o *bloom filter* com a informação recolhida do pacote de controlo (9, 10, 11, 12 e 13). Assim, por cada pacote do ficheiro (14), verifica a sua presença no filtro (15) através do método *contains*. Como tal, se um único bit for igual a zero, significa que o pacote não está no filtro e consequentemente, retorna falso. Caso contrário, retorna *true*, como explicado na subsecção 4.4.3.

Se o pacote está presente no filtro, então avalia se já contém esse mesmo pacote nas suas estruturas internas e no seu armazenamento (17). Se sim, o nó recetor atualiza a sua *vicinity info* (18) e ordena a lista de pacotes a ser enviados (19). Caso contrário, avalia se o pacote pertence a um novo ficheiro (21), e em caso positivo, ativa a sua *flag canSendControlPacket* (23).

Depois disso, o nó recetor verifica se tem nova informação que o nó transmissor não tem, para enviar (24), e se sim, começa a enviar os novos pacotes de dados através da ativação da sua *flag canSendControlPacket* (25).

Se o ficheiro está completo (o número total de pacotes que o ficheiro ocupa é igual ao número de pacotes armazenados) (26), o nó recetor verifica se o ficheiro está ou não no seu armazenamento (27), e se não estiver adiciona informação do novo ficheiro (entre as quais, o número total de pacotes que o nó ocupa) e ativa a *flag canSendControlPacket* (29).

Assim, o processo de receção de uma pacote de controlo é finalizado.

Como referido anteriormente, o *bloom filter* oferece uma representação que reduz o espaço ocupado, com o custo da adição dos falsos positivos. Segundo *Mitzenmacher* [13] existem três métricas fundamentais de desempenho do *Bloom Filter* que podem ser ajustadas entre si: tempo de computação, correspondente ao número de funções de *hash* (k), tamanho, correspondendo ao tamanho da *bitTable* (m), e a probabilidade de erros, que diz respeito à probabilidade de falsos positivos desejada (fp). Assim, de forma a estudar o impacto de

Algorithm 6 Control Packet Reception Bloom Filter

```
1: if received Packet and !bandwidthFull then
2:     Peek number of Files
3:     for file = firstFile to file = lastFile do
4:         Peek fileID
5:         Peek total number of packets
6:         Peek number of packets stored
7:         if File is not completed then
8:             Peek neighbour's parameters from packet's header
9:             Create bloom filter with neighbour's parameters
10:            Peek neighbour's hash functions
11:            Peek neighbour's bit table
12:            Replace calculated hash functions with neighbour's hash functions
13:            Copy neighbour's bit table to bloom filter's bit table
14:            for packet = firstPacket to packet = lastPacket do
15:                Get packet's hash
16:                if Packet is in bloom filter then
17:                    if packet with this hash already in structure and in storage then
18:                        Update vicinity info
19:                        Sort list of data packets to send to neighbours
20:                    else
21:                        if there is a new file to donwload then
22:                            Set canSendControlPacket flag to true
23:            if receiver has different hashes to send then
24:                Set canSendDataPacket flag to true
25:        else
26:            if File doesn't exist in storage then
27:                Add file's info to storage
28:                Set canSendControlPacket flag to true
```

falsos positivos que podem ser tolerados, ou que podem ser reduzidos o suficiente para que a economia de espaço oferecida pelo *bloom filter* compense a probabilidade de erros, foram estudadas e propostas duas implementações de *bloom filter*: uma com tamanho variável e outra com tamanho fixo.

Para tornar possível a implementação dos dois tipos de *bloom filters* propostos, dois sub-módulos foram criados no módulo *Handler*, designados *BFSizeFixed*, para *Bloom Filter* com tamanho fixo, e *BFVariableSize*, para *Bloom Filter* com tamanho variado. A única diferença entre eles é que no *BFSizeFixed* o parâmetro *projectElementsCount* corresponde ao número total de pacotes que um ficheiro ocupa, sendo por isso constante, e consequentemente, conferindo-lhe um tamanho fixo. Pelo contrário, no *BFVariableSize*, o mesmo parâmetro corresponde ao número de pacotes armazenados num determinado momento, sendo por isso variável, tomando valores de zero até ao número total de pacotes que um ficheiro ocupa. Como o cálculo do tamanho da *bitTable* tem em consideração o valor do parâmetro *projectElementsCount*, e como este é variável, confere-lhe, então, a caraterística de tamanho variável.

5.7 Implementação da estratégia LRBFAdvanced

A estratégia proposta, designada LRBFAdvanced, consiste na integração da estratégia *BitArray* com as várias otimizações efetuadas ao nível dos pacotes de dados, de forma a diminuir o problema de *broadcast storm*. A estratégia *BitArray* foi a solução que obteve melhores resultados na redução do tamanho dos pacotes de controlo e consequente diminuição do *overhead* a nível destes pacotes.

5.7.1 Início e Fim da Disseminação

O processo de início e fim da disseminação manteve-se o mesmo que na estratégia LRBF. No início da disseminação, a *flag canSendControlPackets* está desativada nas OBUs e ativa nas RSUs, pois é delas que a informação parte. Assim, apenas as RSUs partilham periodicamente o *bit array* que codifica os pacotes correspondentes a cada ficheiro. Quando uma OBU sem conteúdo está na área de cobertura de uma RSU, recebe o pacote de controlo e assim sabe que a RSU tem conteúdo que ela não tem. Como tal, ativa a *flag canSendControlPackets*, podendo agora enviar periodicamente pacotes de controlo. A RSU recebe um pacote de controlo vazio da OBU, e sabe que ela própria tem conteúdo que a sua vizinha não tem. Logo, a RSU ativa a *flag canSendDataPackets* para permitir o envio de pacotes de dados. Como resultado, o processo de disseminação começa.

Em suma, quando a *flag canSendControlPackets* está ativa permite o envio de pacotes de controlo sempre que o nó tenha nós vizinhos. Já a *flag* canSendDataPackets é responsável pelo envio em *broadcast* de pacotes de dados. No início da disseminação está desativada tanto para as RSUs como para as OBUs.

O processo que diz respeito ao fim da disseminação acontece sempre de duas formas: quando um nó deixa de ter vizinhos e/ou quando todos os seus nós vizinhos contêm o mesmo conteúdo do nó transmissor. Como resultado, a *flag canSendDataPackets* é desativada.

5.7.2 Envio e Receção de Pacotes de Controlo

O processo de envio e receção de pacotes de controlo é descrita na abordagem *BitArray* na secção 5.5. Contudo, foi necessário alterar ligeiramente o algoritmo de receção de pacotes de controlo da abordagem *BitArray* para tornar possível o processo de desmarcação de um pacote como enviado. Assim, sempre que um nó detete que um pacote marcado como enviado não esteja presente num nó vizinho, desmarca esse pacote como enviado. As alterações necessárias são apresentadas a negrito no Algoritmo 7. Já o algoritmo de envio de pacotes de controlo manteve-se igual.

Assim, quando um nó recebe um pacote de controlo, antes de enviar os pacotes de dados que o seu vizinho não tem, desmarca todos esse pacotes como enviados.

5.7.3 Envio e Receção de Pacotes de Dados

De forma a introduzir as melhorias efetuadas ao nível dos pacotes de dados foi necessário alterar o processo de envio e receção destes mesmos pacotes. O processo de envio de pacotes de dados é descrito no Algoritmo 8 e as modificações introduzidas foram: quando um pacote é enviado é marcado como enviado, e sempre que o tamanho da lista com a informação de quantos vizinhos possuem determinado pacote for igual ao número de vizinhos o pacote não é enviado, pois já está presente em todos os vizinhos.

Se o nó transmissor tem apenas OBUs como vizinhos (2) e tem pacotes armazenados (3), então seleciona o primeiro *hash* do pacote da lista de pacotes a transmitir (4). Este *hash* diz respeito ao primeiro pacote a ser selecionado do seu armazenamento, correspondendo ao pacote mais raro. Antes disso, verifica se o pacote ainda não foi enviado (5). Posteriormente verifica se a lista de informações de vizinhos que contém esse pacote tem um tamanho inferior ao número de vizinhos (6), o que significa que o pacote está em falta para algum vizinho. Se o seu tamanho for igual ou superior ao número de vizinhos, significa que todos os vizinhos já têm esse pacote, e como tal, não é necessário enviar. Depois, o pacote correspondente é

Algorithm 7 Control Packet Reception Bit Array in strategy LRBFAdvanced

```
1: if received Packet then
2:     Peek number of Files
3:     for file = firstFile to file = lastFile do
4:         Peek fileID
5:         Peek total packets
6:         Peek packets stored
7:         if File is not completed then
8:             Peek neighbour's bit array
9:             for k_bit = firstK_bit to k_bit = lastK_bit do
10:                GetBit with k_bit
11:                if bit is set then
12:                    Get packet hash with k_bit
13:                    if packet with this hash already in structure and in storage then
14:                        Update vicinity info
15:                        Sort list of data packets to send to neighbours
16:                    else
17:                        if there is a new file to donwload then
18:                            Set canSendControlPacket flag to true
19:            for packet = fist packet to packet = last packet in list of packets to send do
20:                if packet doesn't in neighbour's bit array then
21:                    Set packetHasAlreadySent to false
22:            if receiver has different hashes to send then
23:                Set canSendDataPacket flag to true
24:        else
25:            if File doesn't exist in storage then
26:                Add file's info to storage
27:                Set canSendControlPacket flag to true
```

enviado em *broadcast* (8), sendo marcado como enviado (10), e o seu número de transmissões é incrementado uma unidade (9). A lista de pacotes a enviar é ordenada (11) e é feita a atualização da informação correspondente aos pacotes enviados a ser escrita no ficheiro de *log* (12).

Algorithm 8 Data Packet Forwarding in LRBFAdvanced

```
1: procedure BEGIN
2:     if node's vicinity is only OBUs then
3:         if node has packets then
4:             Get packet hash
5:             if packet hasn't already sent then
6:                 if Size of packet's vicinity Info List < Number of neighbors then
7:                     Update packet's header info
8:                     Send packet in broadcast
9:                     Increment number of tx
10:                    Set packet has already sent
11:                    Sort list of packets to send
12:                    Logging
13:    Sleep
```

De seguida, o algoritmo de receção de pacotes de dados é descrito no Algoritmo 9 e a principal modificação efetuada foi: sempre que um nó recebe e/ou "ouve"um pacote marca esse pacote como enviado.

Como a informação original a ser descarregada da Internet parte das RSUs, apenas as OBUs são consideradas como nós recetores da informação. Assim sendo, se um pacote foi recebido da interface WAVE (2), é verificado se já se encontra no armazenamento do nó (3). Em caso afirmativo, se o pacote for proveniente de uma OBU válida (com RSSI superior a 15), apenas a *vicinity info* é atualizada (5), com a inserção de um novo elemento (com a

necessidade de ordenação da lista de pacotes a enviar), ou se este já estiver presente na lista, atualiza apenas o seu tempo atual. Posteriormente o pacote é marcado como enviado (6). Caso o pacote não exista no seu armazenamento (9), e caso seja proveniente de outra OBU vizinha (11), o pacote é inserido na *vicinity info* (12). O pacote é também inserido na lista de pacotes a enviar. Caso o pacote seja proveniente de uma RSU vizinha (14), apenas é inserido na lista de pacotes a enviar. Em ambos os casos o pacote é marcado como enviado (13 e 15). Se o pacote pertencer a um novo ficheiro (16), a *flag* responsável pelo envio de pacotes de controlo é ativada (17). Por último, o pacote é inserido no armazenamento do nó (19), sendo a informação escrita no ficheiro de *log* (20).

Algorithm 9 Data Packet Reception in LRBFAdvanced

```
 1: procedure BEGIN
 2:     if packet is from WAVE interface then
 3:         if packet already exists then
 4:             if is from a valid OBU neigh then
 5:                 Update vicinity info list
 6:                 Set packet has already sent
 7:                 if inserted a new element in vicinity info then
 8:                     Sort list of packets to send
 9:         else
10:             if is from a valid neigh then
11:                 if is from an OBU then
12:                     Insert packet in vicinity info
13:                     Set packet has already sent
14:                 if is from a RSU then
15:                     Set packet has already sent
16:                 if detected new file to download then
17:                     Set canSendControlPacket flag to true
18:             if packet is not in storage then
19:                 Store packet in storage
20:         Logging
```

5.7.4 Pacotes de dados ouvidos

Foi introduzido um novo algoritmo cujo objetivo é a escuta de pacotes de dados na rede (Algoritmo 10). Assim, sempre que um nó escute um pacote de dados na rede (2), que esteja presente no seu armazenamento (3), marca esse pacote como enviado (4), independentemente se esse pacote poder ser ou não recebido. Para além de marcar o pacote como enviado, atualiza também a lista que indica a informação dos vizinhos que contêm esse pacote (6), ordenando-a, de seguida (8).

Algorithm 10 Overhearing of data packets in LRBFAdvanced

```
1: procedure BEGIN
2:     if packet is from WAVE interface then
3:         if packet already exists then
4:             if is from a valid OBU neigh then
5:                 Set packet has already sent
6:                 Update vicinity info list
7:                 if inserted a new element in vicinity info then
8:                     Sort list of packets to send
9:     Logging
```

5.7.5 Atualização da Lista de Vizinhos

Da mesma forma foi necessário alterar o algoritmo responsável por atualizar a lista de informações de vizinhos correspondente a cada pacote (*packet's vicinity info list*). Este processo é descrito no Algoritmo 11 e a principal modificação foi: sempre que toda a informação dos vizinhos que contêm um determinado pacote for apagada, o pacote é desmarcado como enviado.

Algorithm 11 Refresh thread in LRBFAdvanced

```
 1: procedure BEGIN
 2:     Collect actual_time
 3:     for packet = first packet to packet = last packet of list of packets to send do
 4:         for element = first element to element = last element of vicinity info do
 5:             if actual_time - element time ≥ ELEMENT_VALID_TIME then
 6:                 Erase element from packet's vicinity info
 7:         if packet's vicinity info is empty then
 8:             Set isAllEmpty flag to true
 9:         if at least one element of packet's vicinity info has removed then
10:             Set is2Sort flag to true
11:     if isAllEmpty flag is set to true then
12:         Set canSendDataPackets flag to false
13:         Set packetHasAlreadySent flag to false
14:     if is2Sort flag is set to true then
15:         Sort list of packets to send
16:     Sleep Random time
```

O algoritmo responsável por atualizar a lista de vizinhos que contém cada pacote corresponde ao ciclo de vida de um *thread*, criada exclusivamente para este propósito. Cada elemento desta lista tem um tempo de vida limite igual a ELEMENT_VALID_TIME. Como tal, de forma periódica é verificado o tempo de vida de cada elemento desta lista. Os elementos que tiverem um tempo de vida igual ou superior a ELEMENT_VALID_TIME são apagados.

No início do procedimento é recolhido o tempo atual (2). Para cada pacote da lista de pacotes a enviar (3) é avaliado cada elemento da lista de informações de vizinhos que contém esse pacote (4). Se o tempo atual recolhido subtraído ao tempo do pacote pelo qual foi inserido na lista de informações de vizinhos que contém esse pacote, for igual ou superior a ELEMENT_VALID_TIME (5), o elemento é removido (6). Se pelo menos um elemento desta lista for removido (9) então é necessário ordená-la novamente (10). Se todos os elementos foram removidos (7) então as *flags canSendDataPackets* e *packetHasAlreadySent* são desativadas (12 e 13).

Ao fim deste procedimento, a *thread* responsável adormece por um período aleatório estabelecido entre MIN_REFRESH_PERIOD e MAX_REFRESH_PERIOD, que correspondem a macros definidas inicialmente antes do código ser executado. Os valores que foram estabelecidos são 15 e 30 segundos, respetivamente.

5.8 Integração do código fonte nas placas *NetRider*

O código que é executado no mOVERS é o mesmo que é executado nas placas *NetRider*; no entanto, é necessário fazer a integração para ser executado nas placas.

O processador das placas *NetRider* tem uma arquitetura diferente do qual o mOVERS é executado. O mOVERS foi programado para ser executado em ambiente *Linux*. O sistema operativo que corre nas placas é o *VeniamOS*. O *VeniamOS* é um sistema operativo baseado

em *OpenWRT* [68], mais propriamente, o *VeniamOS* é um *fork* do OpenWrt. O OpenWrt é um sistema operativo baseado em Linux, mas está customizado para ser executado em sistemas embutidos, normalmente *routers*.

Como o código fonte do mOVERS é para ser executado em processadores com arquiteturas diferentes, é necessário fazer a compilação cruzada (do inglês *cross-compiled*), neste caso, para as arquiteturas x86 e MIPS (ou *AR71XX MIPS32*).

A compilação cruzada é feita através do SDK do *VeniamOS*, que acaba por ser o SDK do OpenWrt, sendo disponibilizado em [69]. O SDK deve ser instalado na máquina onde está o código fonte. Devem ser criadas um conjunto de *makefiles* para ajudar na compilação cruzada.

Após a compilação cruzada do código fonte é gerado um ficheiro binário. Este ficheiro binário contém um conjunto de instruções a serem executadas pelo processador com a arquitetura *AR71XX MIPS32*, estando pronto para ser executado nas placas.

Contudo, para que os nós comuniquem uns com os outros, é necessário que estejam sincronizados entre eles. Para isso, existem duas formas de o fazer. A primeira pode ser feita através do uso do GPS, desde que cada nó esteja equipado com uma antena GPS e haja sinal de GPS. Além disso, outra desvantagem em relação ao uso do GPS é o elevado consumo da bateria. A segunda forma é feita através do mecanismo Network Time Protocol (NTP) [70], uma vez que é um protocolo nativo do *OpenWRT*. A solução adotada foi a sincronização com base no mecanismo NTP, uma vez que no cenário de teste em laboratório o sinal de GPS era inexistente.

5.9 Considerações Finais

Neste capítulo é feita uma descrição completa da arquitetura do emulador usado, designado mOVERS. Foram descritas as modificações efetuadas ao nível da largura de banda para incluir, também, os pacotes de controlo, assim como, as modificações necessárias para que a implementação e integração das quatro abordagens de diminuição do tráfego de controlo fosse possível. Além disso, foram também descritas as modificações necessárias para incluir a nova estratégia resultante, LRBFAdvanced, correspondente à melhor abordagem de otimização do tráfego de controlo (*BitArray*), assim como as diversas modificações para diminuição do tráfego de dados.

Por último foram, também, apresentados os principais passos realizados para integrar e executar o código-fonte desenvolvido nas placas *NetRider*.

Capítulo 6

Testes e Resultados

6.1 Descrição do capítulo

Após a integração e implementação das estratégias propostas é necessário proceder à avaliação de cada uma delas, por forma a saber qual a mais eficiente. Assim, este capítulo apresenta o equipamento utilizado, os cenários de teste bem como os resultados obtidos. A organização do capítulo é a seguinte:

- *secção 6.2*: *Equipamento e Software* - descreve as caraterísticas e as especificações do equipamento e *software* usado para os testes e obtenção dos resultados.
- *secção 6.3*: *Scritps de Suporte* - apresenta os *scripts* utilizados para processamento dos dados obtidos durante todas as experiências realizadas.
- *secção 6.4*: *Cenários e Datasets* - apresenta uma descrição detalhada dos cenários avaliados, abrangendo um conjunto de parâmetros como número de nós, área geográfica e períodos de tempo.
- *secção 6.5*: *Resultados das estratégias usando o mOVERS* - apresenta e discute os resultados obtidos através do emulador mOVERS.
- *secção 6.6*: *Resultados obtidos no Laboratório* - apresenta e discute os resultados obtidos nos testes realizados em laboratório.
- *secção 6.7*: *Resultados obtidos nos Moliceiros* - apresenta e discute os resultados obtidos no teste realizado nos moliceiros da ria de Aveiro.
- *secção 6.8* : *Considerações Finais* - apresenta o resumo deste capítulo.

6.2 Equipamento e Software

As soluções propostas foram desenvolvidas num máquina virtual onde o mOVERS é executado, usando o *Eclipse* como IDE. dentro de um servidor. A Tabela 6.1 mostra as especificações do servidor onde está alojada a máquina virtual usada.

O acesso à máquina virtual é feito através de *MobaXterm*. Esta máquina virtual está alojada num servidor que compartilha os seus recursos com outros processos que podem

Processador	Intel®Xeon®CPU E5620 @ 2.40 GHz x8
Memória RAM	12 GB
Sistema Operativo	64-bit Ubuntu 14.04 LTS

Tabela 6.1: Especificações da Máquina Virtual usada para desenvolver as soluções de otimização propostas e executar o emulador mOVERS.

ocorrer. Como tal, todas as métricas de desempenho relativas a esta máquina não devem ser consideradas absolutas, uma vez que estas variam, cada vez que a mesma simulação é realizada.

Para gerar os resultados obtidos para cada solução proposta foi utilizado o MATLAB.

As placas *NetRiders* versão 3 (apresentadas na Figura 6.1) foram usadas quer no teste em laboratório, quer no teste com os moliceiros, e são as mesmas usadas em [71]. A Tabela 6.2 apresenta as principais características destas placas.

Figura 6.1: Placa *NetRider* RSU/OBU [14].

Processador	AR71XX MIPS32
Memória RAM	64MB
Sistema Operativo	VeniamOS
Disco Rígido	128MB
Módulos	IEEE 802.11a/b/g e 802.11p
Portas	Ethernet, serial (RS-232) e USB

Tabela 6.2: Especificações da Placa *NetRider*.

6.3 Scripts de Suporte

Os dados utilizados para construir os resultados obtidos são as informações recolhidas pelo módulo de *Logging* ao longo do tempo das emulações e das experiências, dependendo

do cenário em teste. Os resultados obtidos para a avaliação do desempenho das abordagens para diminuição do tráfego de dados e controlo estão representados em gráficos construídos através do processamento destes dados, utilizando *scripts* de MATLAB [72].

6.4 Cenários e Datasets

A avaliação das soluções de otimização propostas foi feita através da experiência de uma rede veicular real localizada na cidade do Porto [71]. Esta rede veicular real interconeta mais de 600 veículos, sendo estes autocarros públicos, camiões do lixo e veículos municipais.

Dois *datasets* foram disponibilizados, com informações recolhidas sobre a topologia e o comportamento da rede do Porto, ambos de 24 horas, referentes aos dias 31 de Outubro de 2014 (*dataset* 2) e 13 de Fevereiro de 2015 (*dataset* 3). Estes dados consistem em várias informações como as coordenadas GPS ou a lista de nós vizinhos, por cada 5s e 2s, respetivamente,ou seja, apresentam toda a informação de mobilidade e conetividade da rede veicular. No entanto, para analise estatística, apenas o *dataset* mais recente foi escolhido, uma vez que o número de OBUs e RSUs é maior em relação ao *dataset* anterior.

Os *datasets* são compostos por dados recolhidos de todos os nós da rede, sendo filtrados pela posição geográfica ou pelo intervalo de tempo.

A região geográfica da cidade do Porto selecionada para emular a rede veicular real, através do mOVERS, foi o centro da cidade, que corresponde à avenida dos Aliados e tem um raio de 1 km, para ambos os *datasets*. Segundo, *Pessoa* [12] esta região foi selecionada devido à alta densidade de tráfego e mobilidade de veículos (OBUs) e também devido ao número elevado de RSUs. Assim, ambos os *datasets* consideram todas as OBUs e RSUs que estão dentro desta área geográfica. A Figura 6.2 apresenta a região geográfica selecionada, bem como a localização das RSUs para o *dataset* 3 (pontos a vermelho).

Desta forma, dois períodos de disseminação de conteúdos na rede foram considerados: período de *rush hour* e período de *non-rush hour*. O período de *rush hour* ocorre durante a manhã do dia 13 de Fevereiro de 2015, das 6-10 horas e tem como foco analisar o comportamento das soluções propostas para uma elevada quantidade de veículos (hora-de-ponta). O período de *non rush hour* decorre entre as 10-14 horas do mesmo dia, e foca-se em avaliar o impacto das soluções de otimização propostas quando a quantidade de veículos é menor em relação ao período anterior.

Por último, o período de *Parking* ocorre das 20-24 horas, também, do dia 13 de Fevereiro de 2015. Este cenário tem como objetivo avaliar a nova estratégia proposta (LRBFAdvanced) em relação ao congestionamento da rede, quer a nível de pacotes de controlo quer a nível de pacotes de dados. Este período é caraterizado por uma densidade bastante grande de veículos estacionados na mesma área geográfica.

Sendo assim, o número de OBUs para o período de *rush hour* é 161, sendo o número de RSUs 18. Por sua vez, o número de OBUs para o período de *non rush hour* é 133 e 18 RSUs. E por fim, para o período de *Parking*, apenas existe 1 RSU e 120 OBUs.

Além disso, as mensagens são enviadas apenas quando o sinal RSSI é superior ou igual a 15 dBm.

6.4.1 Teste no Laboratório

Os testes no laboratório foram efetuados com o objetivo de comparar a nova estratégia implementada LRBFAdvanced num cenário real.

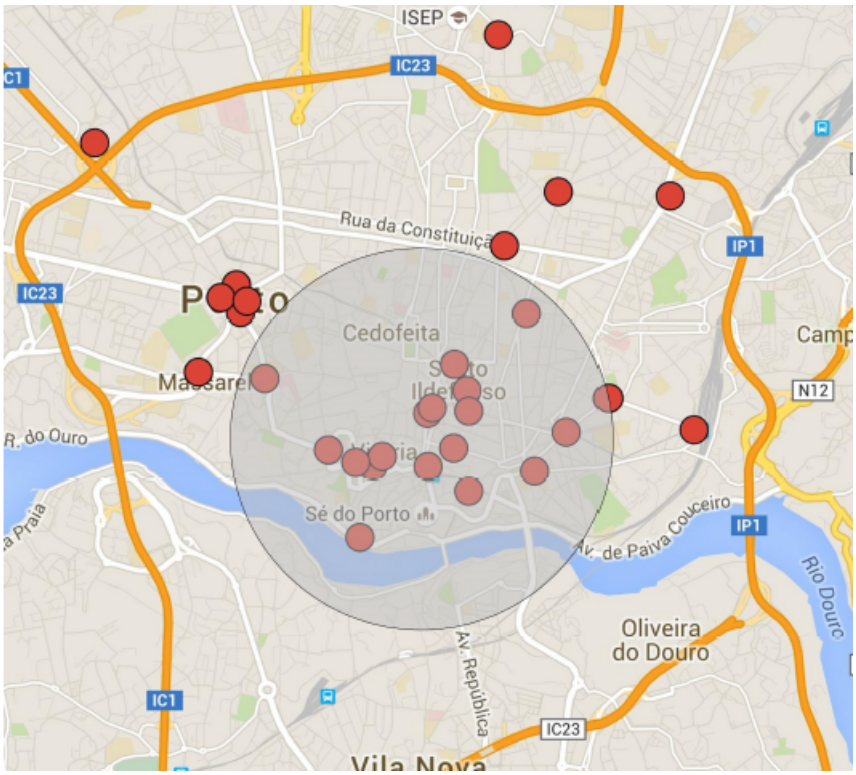


Figura 6.2: Cenário usado para o *dataset* 3 referente ao dia 13 de Fevereiro de 2015 [12].

Uma vez que a estratégia LRBF já foi testada no laboratório num trabalho anterior [12], este teste procurou usar as mesmas condições que o teste anterior, para que a comparação entre as duas estratégias (LRBF vs. LRBFAdvanced) seja mais realista.

A Figura 6.3 apresenta o cenário de teste em laboratório que foi usado. Este cenário é baseado no desenho da planta do edifício 2 do Instituto de Telecomunicações de Aveiro. De igual modo foram usados cinco nós (4 OBUs e 1 RSU), colocados nas mesmas posições em relação ao teste realizado anteriormente. O objetivo inicial deste cenário é de emular a distribuição de conteúdo num cenário real, onde todas as principais caraterísticas das VANET estão presentes, como a conectividade intermitente entre os nós da rede, bem como a existência de um nó, servindo como mula de dados móveis (do inglês *data mule*), correspondente à OBU 645. Esta OBU móvel recolhe partes do ficheiro da infraestrutura fixa, RSU 650, entregando-as, de seguida, aos nós fixos esparsos, OBUs 632 e 656, que não estão na área de cobertura da RSU. A OBU fixa 678 simula o cenário de *parking*, quando existe contacto direto com a infraestrutura fixa.

Como tal, é esperado que nesta experiência, a OBU 645 consiga descarregar completamente o ficheiro da infraestrutura fixa, disseminando-o totalmente pelas OBUs fixas que estão espalhadas pelo cenário. É ainda esperado que a OBU 678 receba o ficheiro mais rápido do que todas as restantes OBUs, uma vez que está sempre na área de cobertura da RSU.

Na Tabela 6.3 está representado o percurso total da OBU 645, com a duração do trajeto percorrido entre cada ponto e a distância correspondente. Dado que é o único nó móvel, toda a experiência se concentra neste nó. A experiência tem início no ponto A, local de onde parte a OBU móvel. Antes de partir, espera 30s, e só depois passa pelos pontos B,C,D,E,B

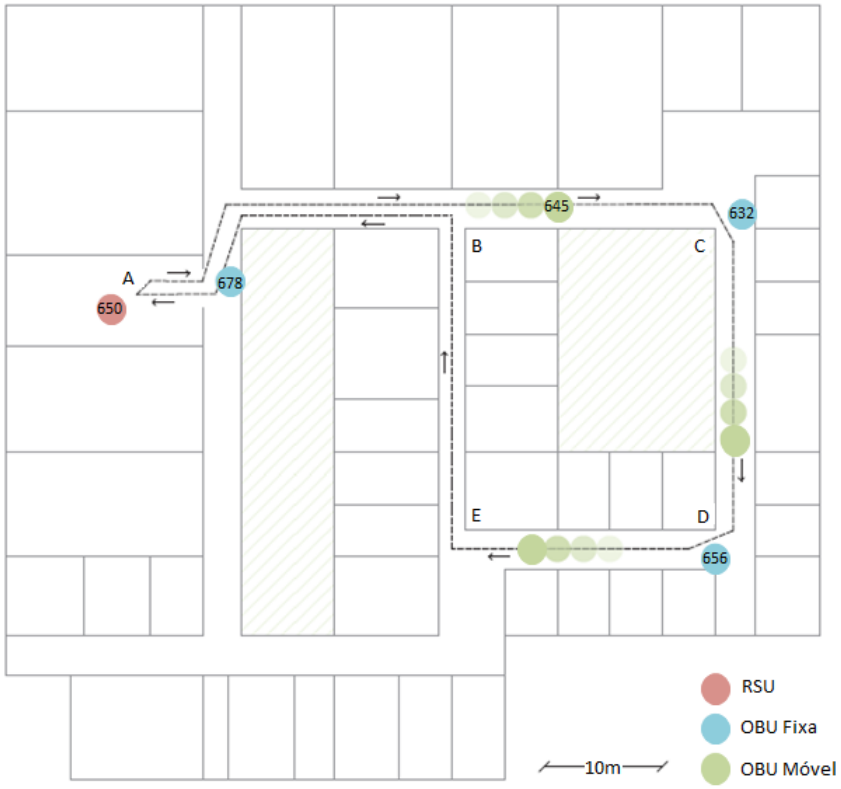


Figura 6.3: Cenário de teste desenvolvido no Laboratório, baseado em [12].

voltando ao ponto A. No ponto A, espera novamente 30s e repete este trajeto mais duas vezes, terminado novamente no ponto A, num total de 360s e uma distância de 462 metros.

6.4.2 Teste nos Moliceiros da Ria de Aveiro

O objetivo do teste cujo cenário são os moliceiros a circular na ria de Aveiro é testar a estratégia LRBFAdvanced numa rede real com conteúdos e utilizadores reais. Neste cenário a experiência não é controlada.

De igual forma à experiência realizada no laboratório foram usadas as mesmas quatro OBUs e a mesma RSU. Duas OBUs foram colocadas em dois moliceiros pertencentes a uma companhia, e as outras duas foram colocadas em outros dois moliceiros de outra companhia (Figura 6.5). Cada companhia tem um total de 4 moliceiros. A RSU foi posicionada à beira da ria ao longo do percurso dos moliceiros, em frente ao fórum de Aveiro (Figura 6.6).

Enquanto que o cenário de laboratório permite ter algum controlo sobre a experiência, isto é, por exemplo sabe-se que a OBU móvel passa pelos pontos A,B,C,D,E,B,A e demora um período de tempo bem definido em cada um deles, no cenário de testes nos moliceiros isto não acontece, pois é um cenário completamente aleatório. O trajeto e a posição dos moliceiros na ria de Aveiro é apresentado na Figura 6.4, bem como a localização da RSU. No entanto, o percurso e a direção que cada moliceiro é decidido pelas companhias, dependendo do número de pessoas para realizar o passeio, no momento, assim como qual o moliceiro que vai efetuar o passeio.

Desde	Para	Tempo (segundos)	Distância (metros)
A	A	30	0
A	B	15	29
B	C	15	23
C	D	15	25
D	E	15	23
E	B	15	25
B	A	15	29
A	A	30	0
A	B	15	29
B	C	15	23
C	D	15	25
D	E	15	23
E	B	15	25
B	A	15	29
A	A	30	0
A	B	15	29
B	C	15	23
C	D	15	25
D	E	15	23
E	B	15	25
B	A	15	29
-	-	360	462

Tabela 6.3: Percurso total percorrido pela OBU 645.

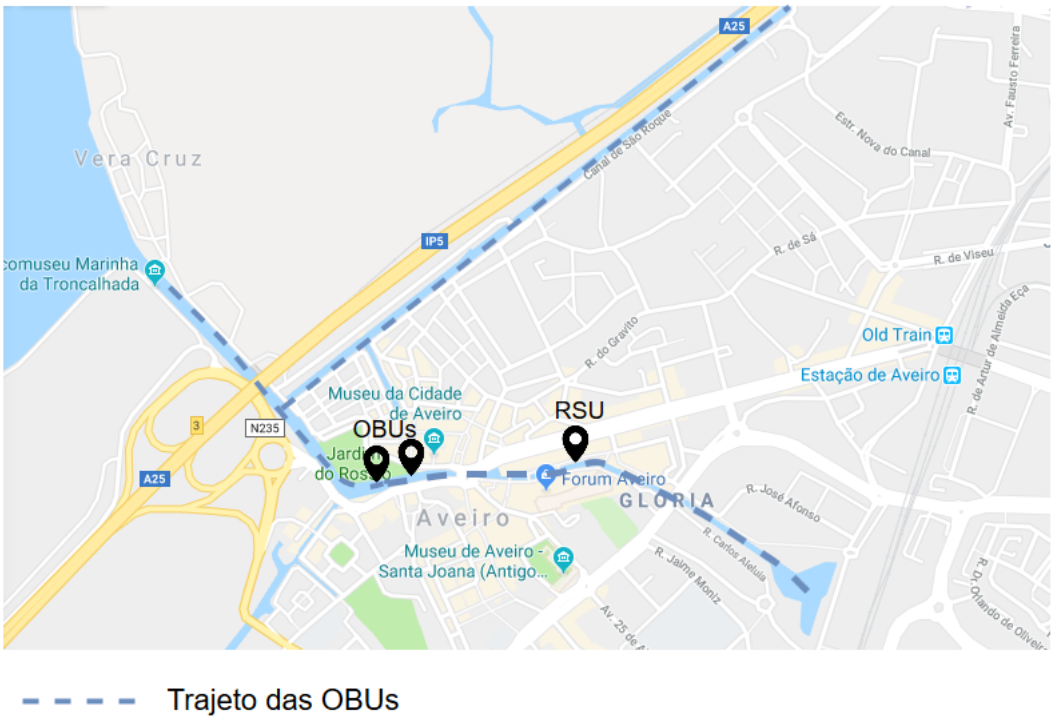


Figura 6.4: Cenário de teste dos moliceiros da ria de Aveiro, percurso efetuado.

Figura 6.5: OBU fixa num dos moliceiro da ria de Aveiro.

6.5 Resultados das estratégias usando o mOVERS

6.5.1 Abordagens com vista a reduzir o tamanho dos pacotes de controlo

As quatro soluções propostas para otimização do tamanho dos pacotes de controlo, diminuindo assim, o *overhead* introduzido na rede, podem ser aplicadas em qualquer estratégia

Figura 6.6: RSU posicionada em frente ao forum de Aveiro.

em que haja troca de pacotes de controlo para informar quais os recursos que cada nó contém. Para demonstrar a eficácia de cada uma das soluções, estas foram implementadas na estratégia mais eficiente, a LRBF, e posteriormente avaliadas utilizando o mOVERS.

Todas os testes consideram a disseminação de um ficheiro de 75MB na rede, dividido em 2256 pacotes de dados, cada um com 32KB.

De forma a considerar o comportamento do emulador próximo do comportamento real deste tipo de serviço na rede real, foi considerada a largura de banda de 1 Mbps, tanto para os pacotes de controlo como para os pacotes de dados transmitidos e recebidos. A largura de banda 1 Mbps é definida como uma pequena percentagem da largura de banda máxima disponível nas placas *NetRider* (27 Mbps) já que o serviço de distribuição de conteúdos não é um serviço de alta prioridade.

De modo a minimizar problemas de desempenho do emulador mOVERS vários procedimentos periódicos não têm um período fixo, pelo que foram definidos com um valor aleatório dentro de um certo intervalo. Como tal, os pacotes de controlo são transmitidos em *broadcast* com uma periodicidade entre 5 e 10 segundos, o período de *refresh* das estruturas internas é entre 15-30 segundos. O período de tempo válido da informação das estruturas internas desde que recebe um pacote de controlo está limitado a 22 segundos.

Contudo, o emulador mOVERS tem um tempo de execução muito elevado. Cada teste de estratégia de disseminação de conteúdo leva 20 horas para ser concluído, tornando mOVERS impraticável para um grande número de testes de experiência de distribuição de conteúdo. O consumo elevado de recursos deve-se ao facto de cada OBU e RSU corresponder à execução de um processo separado dentro do servidor no qual corre o emulador. Como tal, um único servidor tem que executar quase 200 máquinas ao mesmo tempo.

Relativamente à definição dos parâmetros para o *bloom filter* de tamanho fixo, fez-se um ajuste entre o tamanho da *bitTable* (m) e a probabilidade de falsos positivos, de forma a obter

Probabilidade de Falsos Positivos	1%	5%	10%	15%	**20%**	25%
k	7	4	3	3	**2**	2
m	2706	1762	1356	1117	**952**	814
Afetou desempenho dos nós	Não	Não	Não	Não	**Não**	Sim

Tabela 6.4: Configuração dos parâmetros do *Bloom Filter* de tamanho fixo, tendo em conta um ficheiro de 75MB dividido em 2256 pacotes.

uma taxa de falsos positivos aceitável, isto é, de forma a não afetar o desempenho dos nós da rede em completar a sua tarefa (descarregamento completo do ficheiro da rede). Como tal, fez-se variar o valor da probabilidade de falsos positivos no filtro, o que resultou no cálculo de diferentes funções de hash e tamanhos da *bitTable*, como se apresenta a Tabela 6.4. Para isso, foi usado um *script* chamado *bloom_filter_example*, para testar o comportamento adequado de ambas as funções de inserção (*insert*) e contenção (*contains*), bem como a sua resposta.

Relativamente ao valor de *projectElementCount*, este é fixo, o que faz com que o tamanho da *bitTable* seja sempre fixo, ao contrário de *insertedElementCount*, que varia ao longo do teste, como seria de esperar.

Tendo em conta os resultados obtidos na Tabela 6.4, a probabilidade de falsos positivos escolhida foi 20%, uma vez que foi a probabilidade mais baixa (menor tamanho da *bitTable*) que revelou não afetar o desempenho dos nós em descarregar o ficheiro completo da rede. Valores acima desta probabilidade afetam o desempenho dos nós, impedindo-os de completar a sua tarefa.

De forma similar, para o *bloom filter* de tamanho variável, a definição do valor da probabilidade de falsos positivos obteve-se variando os valores de *projectElementCount*, de 1 até 2256 (tamanho total que o ficheiro ocupa em pacotes), até se encontrar um valor de probabilidade aceitável. A probabilidade de falsos positivos escolhida foi $0,01\%$, uma vez que, foi a probabilidade mais baixa que revelou não afetar o desempenho dos nós em descarregar o ficheiro completo da rede. Valores acima desta probabilidade afetam o desempenho dos nós, impedindo-os de completar a sua tarefa.

6.5.1.1 Período de *Rush* e *Non Rush Hour*

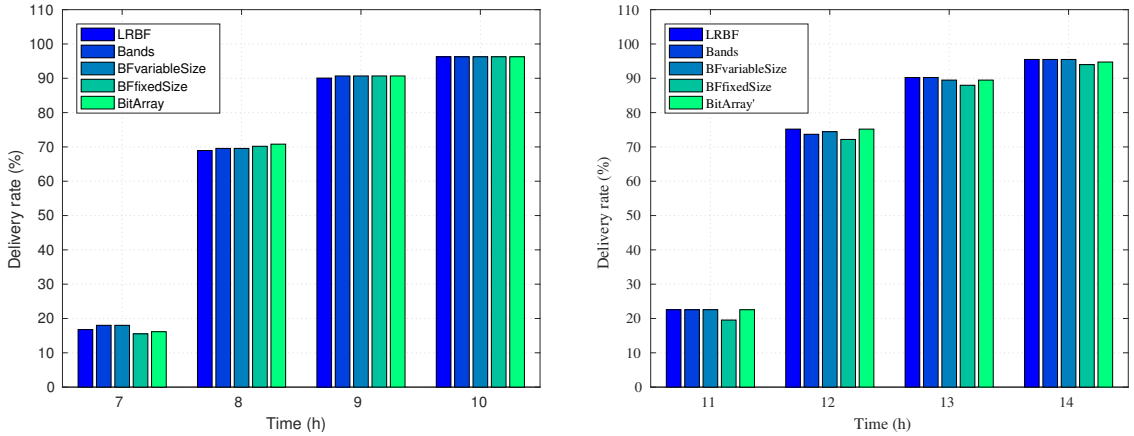
Como referido anteriormente, o período de *Rush* ocorre entre as 6-10 horas e corresponde ao período com uma elevada densidade de veículos (hora-de-ponta), tornando-se assim, o período mais importante para realizar testes. No período de *Non Rush Hour*, a quantidade de veículos é menor e, consequentemente, existe um menor número de contactos. Este período corresponde ao intervalo de tempo entre as 10-14 horas.

As subsecções seguintes apresentam várias métricas usadas para verificar a eficácia das soluções propostas em comparação com a estratégia padrão usada.

Taxa de Entrega

Esta métrica considera a percentagem de OBUs emuladas que descarregaram com sucesso o ficheiro em disseminação, a cada hora da experiência.

A Figura 6.7a mostra que a taxa de entrega aumenta ao longo do tempo até 96.27%, para o período de *Rush Hour*, e que os valores se mantêm os mesmos, para todas as abordagens propostas. A Figura 6.7b mostra também que a taxa de entrega aumenta ao longo do tempo,



(a) *Rush Hour*. (b) *Non Rush Hour*.

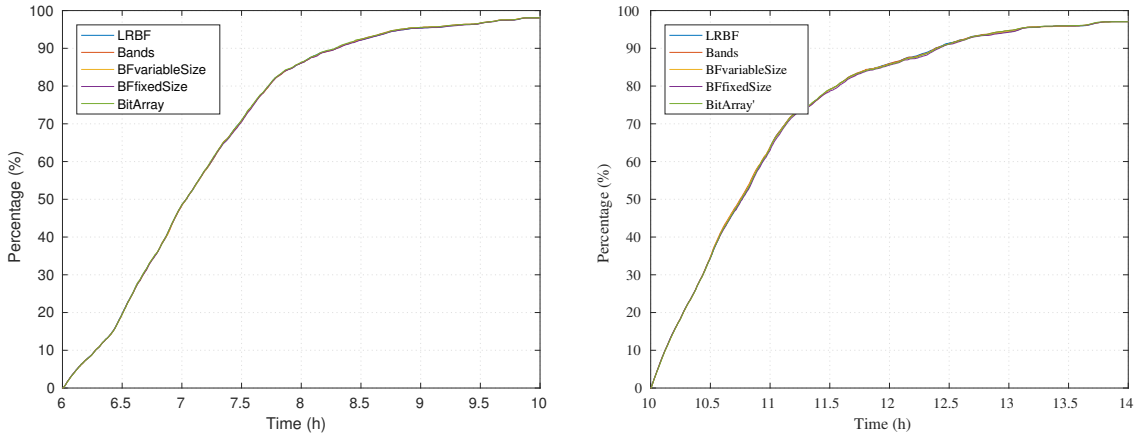
Figura 6.7: Taxa de Entrega: mOVERS, *Rush* e *Non Rush Hour*, estratégias de diminuição do tamanho dos pacotes de controlo.

---

para o período de *Non Rush Hour*, atingindo cerca de 95.5% no final da experiência. Os resultados obtidos em ambos os períodos demonstram que as abordagens de otimização dos pacotes de controlo não causaram qualquer diminuição no desempenho da estratégia padrão.

Distribuição do Ficheiro

Esta métrica diz respeito à percentagem acumulativa do ficheiro distribuído na rede, ao longo da experiência, para todas as OBUs.



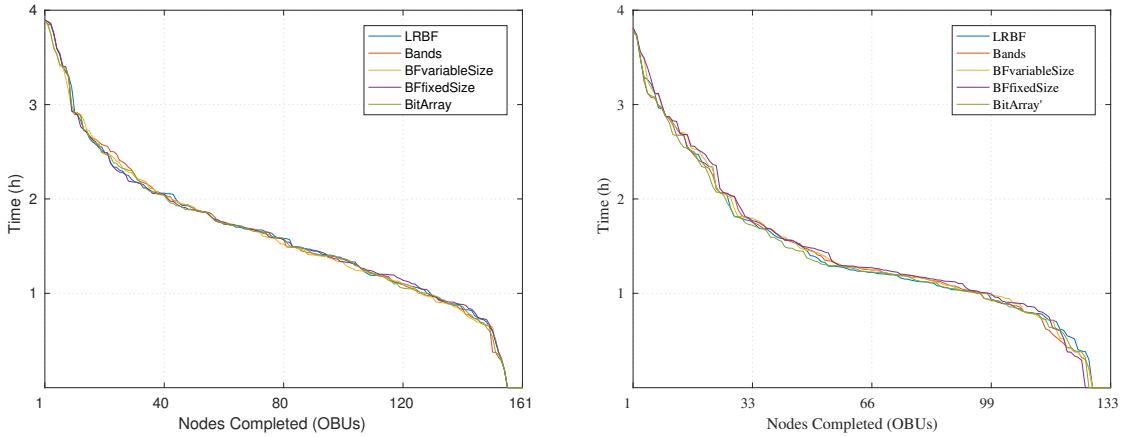
(a) *Rush Hour*. (b) *Non Rush Hour*.

Figura 6.8: Distribuição do Ficheiro: mOVERS, *Rush* e *Non Rush Hour*, estratégias de diminuição do tamanho dos pacotes de controlo.

A Figura 6.8a apresenta os resultados obtidos para o período de *Rush Hour*, sendo que a Figura6.8b apresenta os resultados obtidos para o período de *Non Rush Hour*. É possível concluir que, em ambos os casos, quer para a estratégia padrão quer para as abordagens de otimização, os resultados são os mesmos, atingindo altas percentagens.

End-to-End delay

Esta métrica diz respeito ao tempo que cada nó (OBU) leva para receber o ficheiro completo, desde o início e o fim da experiência. Apenas são considerados os nós que receberam o ficheiro completo.



(a) *Rush Hour*. (b) *Non Rush Hour*.

Figura 6.9: End-to-End delay: mOVERS, *Rush* e *Non Rush Hour*, estratégias de diminuição do tamanho dos pacotes de controlo.

A Figura 6.9a apresenta os resultados obtidos para esta métrica, para o período de *Rush Hour*. Nas primeiras duas horas da disseminação, o número de OBUs que completaram o ficheiro é menor que 25%. No final da terceira hora este número aumenta para cerca de 75%, correspondendo a 120 OBUs. No final da experiência quase todas as 161 OBUs completaram o ficheiro. Note-se que as várias abordagens apresentam um comportamento semelhante, não diminuindo o desempenho da estratégia padrão.

Os resultados do período *Non Rush Hour*, apresentados na Figura 6.9b, apresentam um comportamento semelhante ao período de *Rush Hour*.

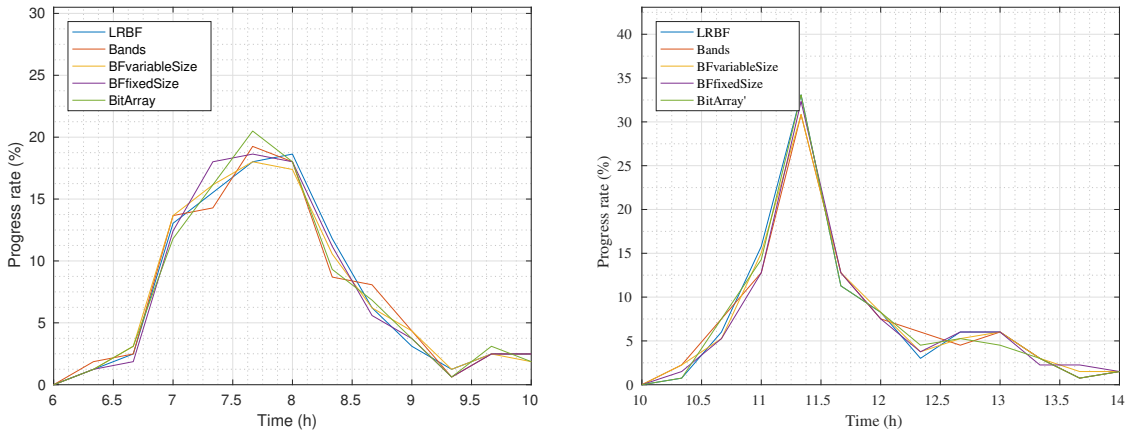
Progress rate

Esta métrica determina a velocidade com que todo o ficheiro está a ser disseminado na rede, a cada 30 minutos. O objetivo é dar uma perspetiva sobre o quanto rápida é a entrega do ficheiro na rede.

A Figura 6.10a apresenta os resultados obtidos para o período de *Rush Hour*. É possível concluir que o comportamento, tanto para estratégia padrão como para as abordagens de otimização implementadas, é bastante semelhante. No entanto, é possível ver observar que, no intervalo de tempo $7.5 - 8$h, a abordagem *BitArray* atinge um valor de pico de cerca de $20,5\%$, sobressaindo em relação à estratégia padrão e às outras abordagens, cujo valor é um pouco mais baixo.

Este resultados demonstram que o período de maior disseminação é feito na segunda hora da experiência, provocando depois uma descida, uma vez que a maioria das OBUs já tem o ficheiro completo.

Os resultados obtidos para o período de *Non Rush Hour* são apresentados na Figura 6.10b. O período de maior disseminação ocorre entre as 11-12h. As abordagens propostas têm exatamente o mesmo comportamento, sem diminuir o desempenho da estratégia LRBF. Em



(a) *Rush Hour*. (b) *Non Rush Hour*.

Figura 6.10: Progress rate: mOVERS, *Rush* e *Non Rush Hour*, estratégias de diminuição do tamanho dos pacotes de controlo.

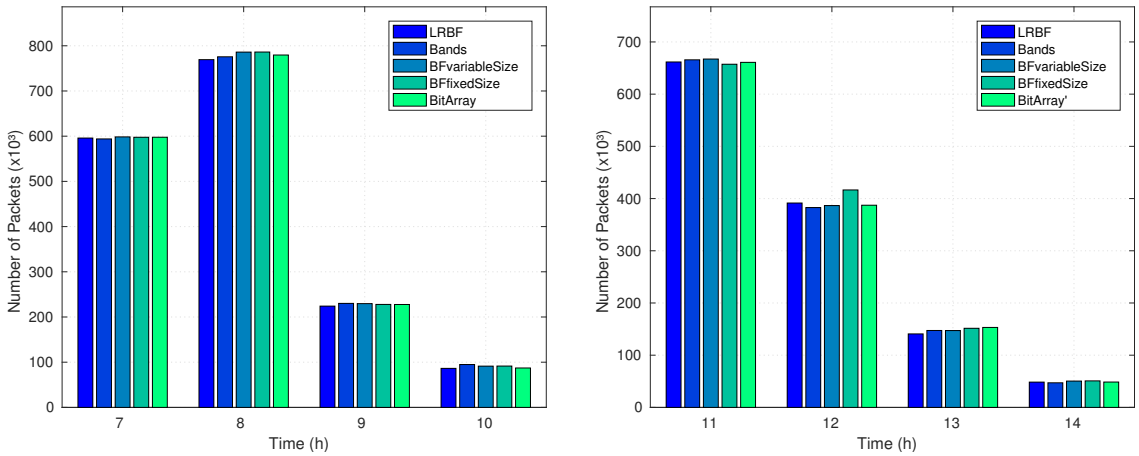
---

comparação com o período de *Rush Hour*, o intervalo de tempo em que ocorre a maior disseminação é maior, uma vez que existe um número menor de contactos.

Ambos os resultados mostram coerência com os resultados apresentados para as métricas anteriores.

Pacotes na rede

Esta métrica diz respeito aos pacotes de dados "ouvidos"na rede a cada hora. É avaliada através da soma acumulativa de todos os pacotes de dados "ouvidos"pelas OBUs na rede, podendo ser interpretada como um medida de congestionamento do meio. A Figura 6.11a mostra os resultados obtidos para o período de *Rush Hour*.



(a) *Rush Hour*. (b) *Non Rush Hour*.

Figura 6.11: Pacotes na rede: mOVERS, *Rush* e *Non Rush Hour*, estratégias de diminuição do tamanho dos pacotes de controlo.

Como tal, regista-se um aumento significativo da quantidade de pacotes de dados "ou-

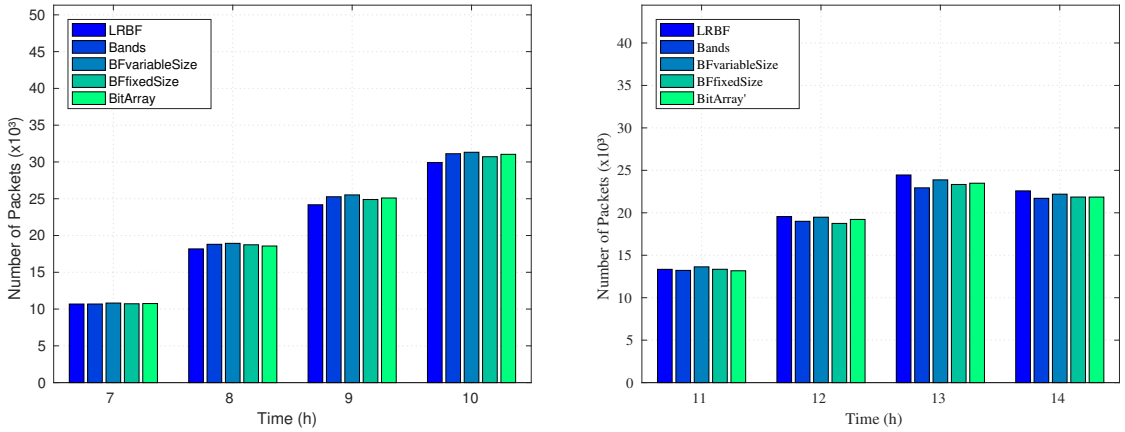
vidos"nas primeiras duas horas da experiência, uma vez que é neste período que existe um maior número de contactos entre os nós da rede e, consequentemente uma maior disseminação do ficheiro, decrescendo bastante na segunda metade da experiência. Este comportamento é explicado pelo facto de a transmissão de pacotes de dados decrescer quando a maioria dos nós vizinhos já receberam a totalidade do ficheiro. Assim, um nó não envia qualquer pacote de dados uma vez que a sua vizinhança já tem todos os pacotes necessários.

A Figura 6.11b apresenta os resultados obtidos para o período de *Non Rush Hour*. É possível verificar que o número de pacotes "ouvidos"na rede decresce ao longo da experiência, uma vez que as OBUs apenas enviam pacotes se os seus nós vizinhos necessitarem. O número de pacotes "ouvidos"na rede decresce com o aumento da taxa de entrega (ver Figura 6.7b). Como o maior período em que a disseminação ocorre é na primeira hora da experiência, então é possível verificar que é nesta hora onde ocorre um pico de pacotes "ouvidos"na rede.

Quer para o período de *Rush* como para o período de *Non Rush Hour*, é possível observar que praticamente todas as abordagens mostram comportamentos semelhantes à estratégia LRBF.

Número de pacotes de controlo transmitidos

Esta métrica corresponde ao número de pacotes de controlo transmitidos a cada hora da experiência. O objetivo desta métrica é avaliar o impacto dos pacotes de controlo na rede. A Figura 6.12a apresenta os resultados obtidos no período de *Rush Hour*.



(a) *Rush Hour*. (b) *Non Rush Hour*.

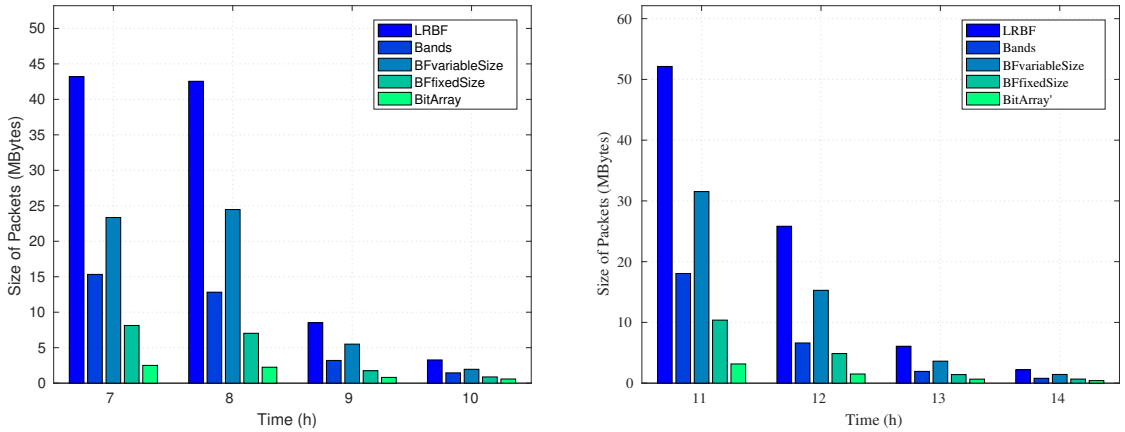
Figura 6.12: Número de pacotes de controlo transmitidos: mOVERS, *Rush* e *Non Rush Hour*, estratégias de diminuição do tamanho dos pacotes de controlo.

Um aumento no número de pacotes de controlo transmitidos é visível ao longo do tempo da experiência. Este comportamento era esperado, uma vez que, quanto mais nós estiverem cientes do conteúdo em disseminação, maior o número de anúncios na rede. É possível verificar que todas as abordagens têm o mesmo comportamento.

A Figura 6.12b apresenta os resultados obtidos no período de *Non Rush Hour*. De igual forma ao período de *Rush Hour*, todas as abordagens propostas mantêm comportamentos similares à estratégia padrão.

Tamanho dos pacotes de controlo

Este métrica diz respeito ao tamanho total dos pacotes de controlo transmitidos, sendo uma das métricas mais importantes para avaliar o impacto e a eficácia das abordagens propostas.



(a) *Rush Hour*. (b) *Non Rush Hour*.

Figura 6.13: Tamanho dos pacotes de controlo: mOVERS, *Rush* e *Non Rush Hour*, estratégias de diminuição do tamanho dos pacotes de controlo.

A Figura 6.13a apresenta os resultados obtidos para o tamanho total dos pacotes de controlo na rede, por hora, para o período de *Rush Hour*. Nas primeiras duas horas da experiência, a estratégia padrão (LRBF) introduz quase 90 MB de *overhead* na rede (Tabela (6.5)), pelo que é um valor muito elevado, quando se trata da disseminação de apenas um ficheiro de 75 MB na rede veicular. Contudo, este valor decresce na segunda metade da experiência, atingindo cerca de 14 MB. Tendo em conta os resultados apresentados na Figura 6.9a, nas duas primeiras horas apenas 25% das OBUs receberam o ficheiro completo, mas, a partir da terceira hora da experiência, o número de OBUs que recebeu o ficheiro completo aumentou significativamente para 75%, e na última hora para 98%. Este decréscimo do tamanho total dos pacotes de controlo, a partir da terceira hora da experiência, foi devido ao aumento do número de OBUs com o ficheiro completo. Quando um nó ainda não recebeu o ficheiro completo, vai anunciar cada *hash* do pacote que contém desse mesmo ficheiro. Para mapear cada *hash* no pacote de controlo são precisos 4 bytes. No entanto, quando uma OBU recebe o ficheiro completo, apenas envia um pacote de controlo de tamanho reduzido, anunciando apenas a informação geral sobre o ficheiro em disseminação (número total de pacotes e a identificação do ficheiro).

As abordagens referentes às *Bandas* e *BFvariableSize* tiveram um comportamento similar à estratégia padrão, uma vez que o tamanho dos seus pacotes de controlo também é variável. Contudo, estas duas abordagens reduzem bastante o *overhead* introduzido na rede. Analisando os valores apresentados pela Tabela 6.5, é possível concluir que a estratégia *Bandas* apenas usa 37% da quantidade de *overhead* da estratégia padrão, para a primeira hora da experiência. Para a segunda hora usa apenas 30%, 37% na terceira hora, e por fim, apenas 33% na última hora da experiência. Como tal, a abordagem *Bandas* apenas usa 35% do tamanho dos bytes usados pela estratégia padrão.

A estratégia *BFvariableSize* apenas usa 50% da quantidade de *overhead* da estratégia padrão.

As abordagens *BitArray* e *BFfixedSize* têm um comportamento constante nas duas primeiras horas, uma vez que o tamanho dos seus pacotes de controlo é fixo. Como a OBU envia um pacote de tamanho reduzido anunciando apenas o perfil do ficheiro em disseminação, este facto explica a variação entre os valores da primeira e segunda horas, e a diminuição no tamanho total dos pacotes de controlo nas duas últimas horas da experiência. A Tabela 6.5 apresenta os valores exatos do tamanho total dos pacotes de controlo para cada hora da experiência, para o período de *Rush Hour*. Como tal, é possível concluir que a abordagem *BFfixedSize*, apenas usa 20% do *overhead* de controlo da estratégia padrão, assim como, a abordagem *BitArray*, que apenas usa 10%. Estas duas abordagens reduzem significativamente a sobrecarga introduzida na rede.

Como visão geral, embora haja variação do tamanho dos pacotes de controlo, a abordagem *BFvariableSize* conseguiu reduzir o *overhead* da rede em 41%. Da mesma forma, a abordagem *Bandas* conseguiu reduzir o *overhead* em mais de 50%. Já a abordagem de *bloom filter* de tamanho fixo conseguiram reduzir mais de 80% o *overhead* introduzido por estes pacotes. A abordagem que claramente se tornou mais eficiente foi *BitArray*, tendo reduzido o tamanho total dos pacotes de controlo em 90%.

Hour	LRBF	Bands	BFvariableSize	BFfixedSize	BitArray
7	43.2020	15.3314	23.3513	8.1314	2.5046
8	42.5496	12.8244	24.4683	7.0306	2.2486
9	8.5209	3.2049	5.5023	1.7522	0.8123
10	3.2758	1.4553	1.9558	0.8797	0.5868

Tabela 6.5: Tamanho Total dos Pacotes de Controlo por hora, para o período de *Rush Hour*.

A Figura 6.13b apresenta os resultados obtidos no período de *Non Rush Hour*. Neste período verifica-se que o comportamento dos valores é algo semelhante ao comportamento observado para o período anterior (*Rush Hour*). As razões para este facto são as mesmas apontadas no período de *rush hour*. Assim, é possível observar que no início da experiência, a estratégia padrão adiciona um congestionamento elevado na rede, diminuindo ao longo do tempo da experiência, considerando que a estratégia converge em termos de taxa de entrega.

Da mesma forma, este decréscimo é explicado pelos resultados apresentado nas Figuras 6.7b, 6.9b e 6.10a. Visto que a entrega do ficheiro acontece entre o intervalo de tempo $11 - 11.6$h, ou seja, é neste intervalo que a maioria dos nós da rede completa o ficheiro, a quantidade de informação a enviar nos pacotes de controlo, também diminui, diminuindo o seu tamanho.

As abordagens de tamanho variado, *Bandas* e *BFvariableSize*, têm um comportamento semelhante à estratégia padrão, da mesma forma que para o período de *rush hour*, reduzindo bastante o congestionamento causado por estes pacotes.

As abordagens com o tamanho fixo dos pacotes de controlo, *BFfixedSize* e *BitArray*, tendem a ter um comportamento mais uniforme. A diminuição do seu tamanho é explicada pelas mesmas razões. Estas duas abordagens reduzem significativamente o *overhead* introduzido.

A Tabela 6.6 mostra detalhadamente os valores do tamanho total dos pacotes de controlo, de todas as abordagens, para cada hora da experiência, para o período de *Non Rush Hour*.

De igual forma, é possível concluir que a abordagem *Bandas* reduz o tamanho total do pacotes de controlo em 64%, a abordagem *BFvariableSize* conseguiu reduzir o tamanho total

destes pacotes em 39% usando 61% dos bytes usados pela estratégia padrão, e as abordagens de tamanho fixo foram as que obtiveram melhor eficácia, tendo reduzido o tamanho total em cerca de 77% e 89% (*BFfixedSize* e *BitArray*, respetivamente). A abordagem que se destacou foi *BitArray*, tendo usado apenas 6% do *overhead* de controlo com a abordagem padrão, para as duas primeiras horas da experiência. Para a terceira e última hora, apenas necessitou de 11% e 19%, respetivamente.

Hour	LRBF	Bands	BFvariableSize	BFfixedSize	BitArray
11	52.1328	18.0552	31.5315	10.3789	3.1640
12	25.8297	6.6179	15.2831	4.8720	1.5072
13	6.0765	1.9415	3.6224	1.4081	0.6631
14	2.1961	0.8026	1.4349	0.6750	0.4152

Tabela 6.6: Tamanho Total dos Pacotes de Controlo por hora, para o período de *Non Rush Hour*.

Em suma, com base nos resultados obtidos, verificou-se que a abordagem *BitArray* obteve os melhores resultados na redução do *overhead* dos pacotes de controlo na rede. Como tal, a abordagem que irá ser integrada na nova estratégia posposta (LRBFAdvanced) irá ser a abordagem *BitArray*.

6.5.2 Abordagem com vista a controlar o *overhead* introduzido pelos pacotes de dados - LRBFAdvanced

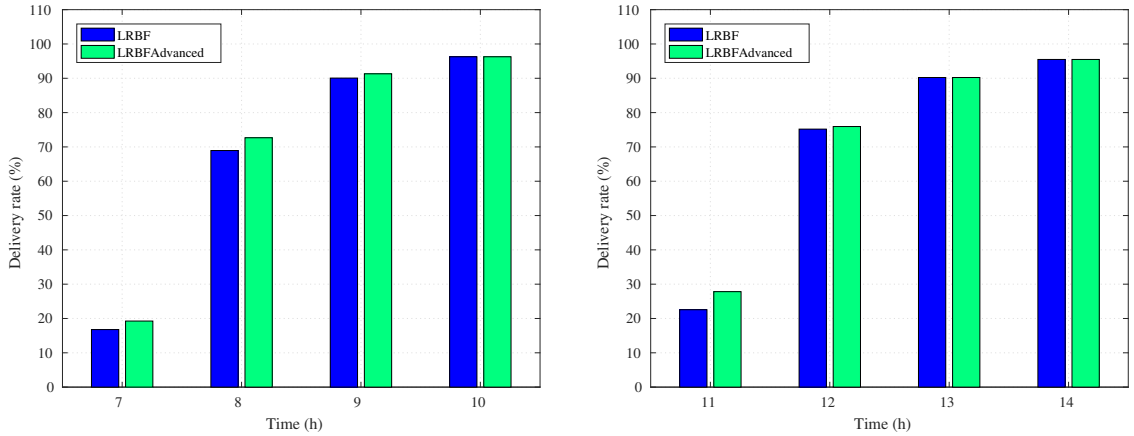
O ficheiro considerado para disseminação foi o mesmo para todos os cenários, assim como as definições de transmissão periódica de pacotes de controlo, o período de *refresh* das estruturas internas, bem como o tempo válido da informação nestas estruturas.

Para melhor demonstrar a eficácia da nova estratégia implementada LRBFAdvanced, houve a necessidade de desenvolver uma nova métrica para avaliar a quantidade de pacotes de dados transmitidos ao longo da experiência.

6.5.2.1 Período de *Rush* e *Non Rush Hour*

Taxa de Entrega

A taxa de entrega do ficheiro ao longo da experiência é apresentada na Figura 6.14a para o período de *Rush Hour*, e na Figura 6.14b para o período de *Non Rush Hour*. Em ambos os períodos, a estratégia LRBFAdvanced é ligeiramente mais rápida na entrega de conteúdo. Este facto deve-se às várias otimizações efetuadas para diminuir o **overhead** de dados, o que leva a um melhor aproveitamento da largura de banda disponível. No entanto, a estratégia LRBF acaba por recuperar na segunda metade da experiência. Para o período de *Rush Hour* a taxa de entrega atinge 96.27%, e 95.49% para o período de *Non Rush Hour*. Os restantes 3.73% para o período de *Rush Hour* e 4.51% para o período de *Non Rush Hour* devem-se a que se considerem OBUs que não estão ativas o tempo todo da experiência, como por exemplo, OBUs que entram e saem da região considerada para experiência (ver secção 6.4). Para futuros testes, deve ser feita uma filtragem de quais as OBUs que permanecem o tempo todo na região considerada para experiência.

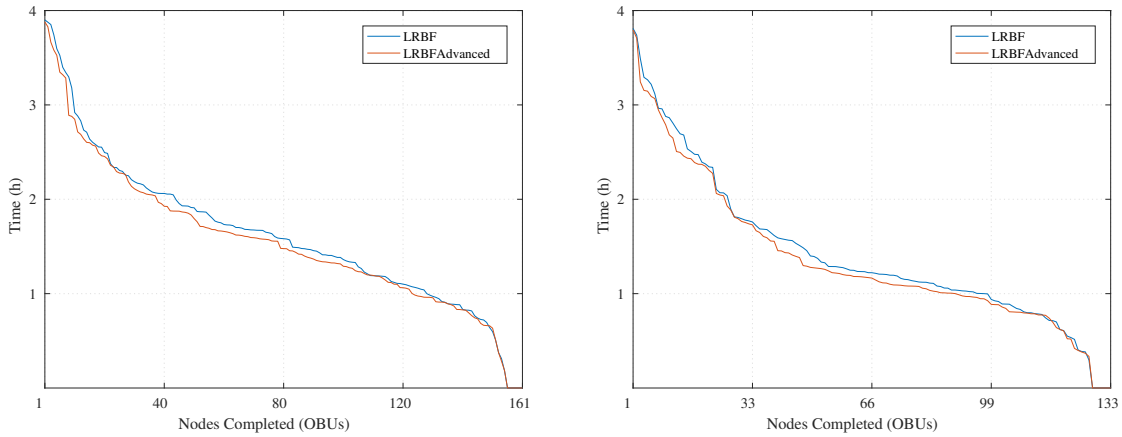


(a) *Rush Hour*. (b) *Non Rush Hour*.

Figura 6.14: Taxa de Entrega: mOVERS, *Rush* e *Non Rush Hour*, LRBF e LRBFAdvanced.

End-to-End Delay

As Figuras 6.15a e 6.15b apresentam o número de OBUs que receberam o ficheiro para o período de *Rush* e *Non Rush Hour*, respetivamente. Em ambos os períodos, o comportamento é muito semelhante para ambas as estratégias. Embora a estratégia LRBFAdvanced seja ligeiramente mais rápida na entrega de conteúdo (Figura 6.14b), este comportamento não é evidenciado nesta métrica. O número de OBUs com o ficheiro completo é o mesmo para ambas as estratégias.



(a) *Rush Hour*. (b) *Non Rush Hour*.

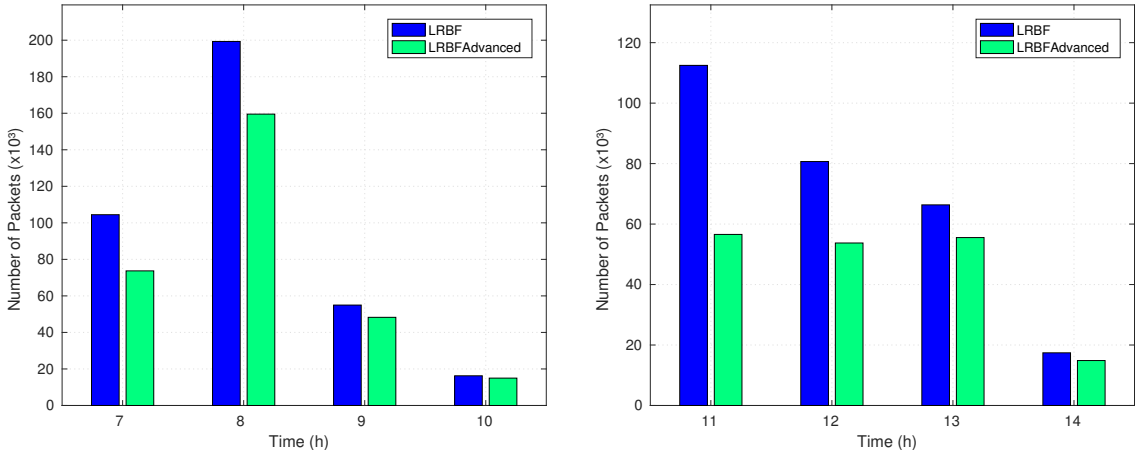
Figura 6.15: End-to-End Delay: mOVERS, *Rush* e *Non Rush Hour*, LRBF e LRBFAdvanced.

Número de Pacotes de Dados Transmitidos

Na Figura 6.16a é apresentado o número de pacotes enviados ao longo das quatro horas da experiência, para o período de *Rush Hour*. É possível verificar que a estratégia LRBFAdvanced enviou menos cerca de 21% dos pacotes de dados transmitidos na estratégia LRBF. Esta diminuição é observada nas duas primeiras horas da experiência, uma vez que é neste intervalo de tempo que ocorre uma maior disseminação do conteúdo do ficheiro.

No período de *Non Rush Hour*, ilustrado pela Figura 6.16b, a diminuição é ainda maior, constituindo cerca de 35% do total dos pacotes enviados pela estratégia LRBF. Esta diminuição é evidenciada na primeira hora da experiência, para cerca de metade, uma vez que a maior disseminação do conteúdo do ficheiro ocorre das 11-12h. Nas horas seguintes ainda é possível observar esta diminuição, sendo que na última hora o envio de pacotes é ligeiramente menor para a estratégia LRBFAdvanced. Neste período, a quantidade de pacotes de dados enviados é menor quando comparado com a quantidade enviada no período de *Rush Hour*. Esta diminuição é esperada, pois no período de *Non Rush Hour*, o número de contactos entre os nós é menor. Como os pacotes de dados são marcados como enviados quando são transmitidos e quando são recebidos e/ou "ouvidos"pelos nós da rede, o *overhead* de dados diminui.

Esta métrica prova que as otimizações para controlar o problema de congestionamento de pacotes de dados tiveram sucesso.



(a) *Rush Hour*. (b) *Non Rush Hour*.

Figura 6.16: Número de Pacotes de Dados Transmitidos: mOVERS, *Rush* e *Non Rush Hour*, LRBF e LRBFAdvanced.

As Tabelas 6.7 e 6.8 apresentam os valores exatos para o período de *Rush* e *Non Rush Hour*, respetivamente.

Hora	**LRBF**	**LRBFAdvanced**
7	104.4680	73.7010
8	199.3220	159.5310
9	54.9790	48.2680
10	16.2680	14.9840

Tabela 6.7: Número de pacotes de dados enviados durante cada hora de experiência, para as estratégias LRBF e LRBFAdvanced, para o período de *Rush hour*.

Pacotes na Rede

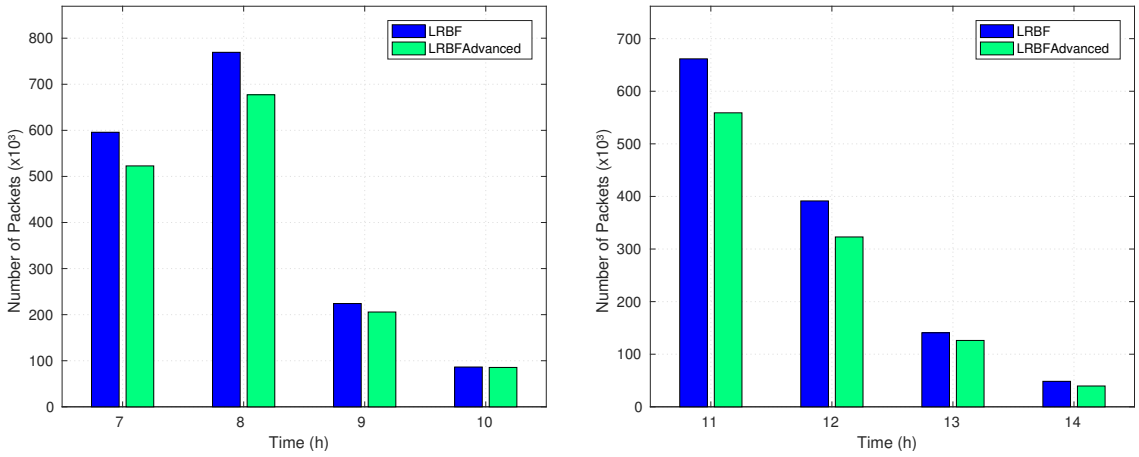
A Figura 6.17a ilustra os pacotes "ouvidos"na rede no período de *Rush Hour*. Uma vez

Hora	LRBF	LRBFAdvanced
11	112.4880	56.5760
12	80.6850	53.7420
13	66.3570	55.5380
14	17.4240	14.8650

Tabela 6.8: Número de pacotes de dados enviados durante cada hora de experiência, para as estratégias LRBF e LRBFAdvanced, para o período de *Non Rush hour*.

que a estratégia envia menos pacotes de dados é natural que haja uma diminuição dos pacotes "ouvidos"na rede. Esta diminuição é notória nas duas primeiras horas da experiência, uma vez que é neste intervalo de tempo que ocorre a diminuição do envio dos pacotes de dados, mostrando que os resultados estão em concordância com a métrica anterior.

No período de *Non Rush Hour*, ilustrado pela Figura 6.17b, o comportamento é o mesmo.



(a) *Rush Hour*. (b) *Non Rush Hour*.

Figura 6.17: Pacotes na Rede: mOVERS, *Rush* e *Non Rush Hour*, LRBF e LRBFAdvanced.

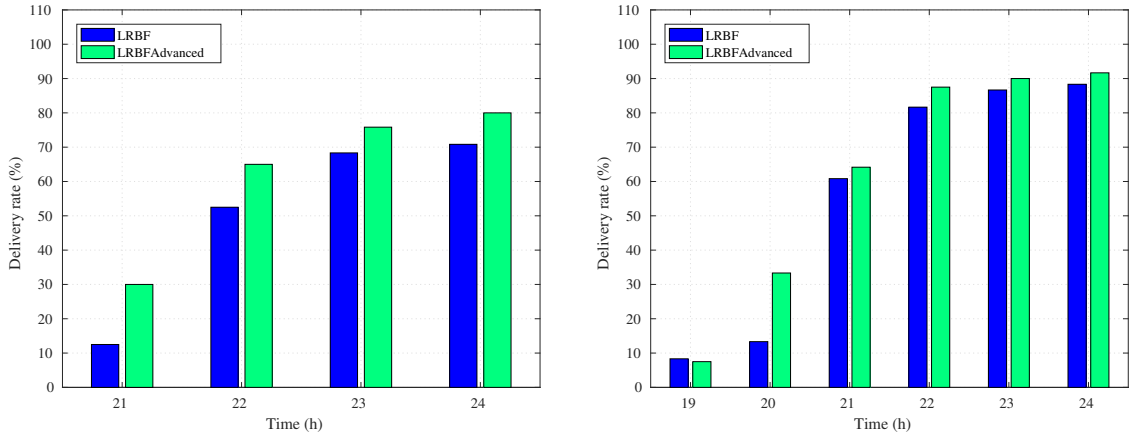
6.5.2.2 Período de *Parking*

O período de *Parking* é bem diferente dos dois períodos anteriores. Diz respeito ao período de tempo em que a grande maioria das OBUs (autocarros) chegam à central de camionagem, terminando o seu percurso. Neste período é onde existe um maior número de contactos entre nós, dado que todos eles terminam o seu percurso neste lugar. Quando os autocarros chegam à central de camionagem, estacionam e permanecem ativos (com o motor ligado) apenas por algum tempo depois.

Segundo Pessoa [12], não foi possível executar as emulações no mOVERS, devido à grande quantidade de nós tentando comunicar entre eles, em *broadcast*, o que produz uma grande quantidade de mensagens internas do mOVERS (mensagens ZMQ), quer para a comunicação de controlo quer para a comunicação de dados. No entanto, com as alterações efetuadas à largura de banda, descritas no na secção 5.3.2, foi possível executar as emulações para este período. Os resultados obtidos são apresentados em baixo:

Taxa de Entrega

A Figura 6.18a ilustra os resultados obtidos para a taxa de entrega relativa ao período de *Parking*. Este período engloba o horário entre as 20-24 horas. É possível verificar que o maior número de contacto entre os nós é entre as 20-21 horas, dado que existe uma maior evolução da taxa de entrega, ou seja, é entre este intervalo de tempo que existe uma maior afluência na chegada de autocarros à estação de camionagem. A Tabela 6.9 apresenta os valores reais da taxa de entrega para o período de *Parking*. A estratégia LRBFAdvanced atinge uma taxa de entrega no final da experiência de 80%, sobressaindo-se em relação à estratégia LRBF, que é de 70%. Contudo, em ambas as estratégias, a taxa de entrega é baixa, uma vez que só 86 OBUs completam o ficheiro na estratégia LRBF e 96 na estratégia LRBFAdvanced. Este facto é explicado pelo período que os autocarros permanecem ativos depois de estacionarem, sendo em muitos dos casos inferior a 20 minutos. Como tal, foi necessário antecipar este período por mais duas horas, começando não às 20 horas mas sim às 18 horas. A Figura 6.18b apresenta os resultados obtidos para o período de *Parking Extended*. Neste período alargado é observado que em ambas as estratégias, a taxa de entrega é maior que o período normal, atingiu 91.67% para a LRBFAdvanced, e 88.33% para a LRBF, conforme apresentado na Tabela 6.10.



(a) *Parking*.

(b) *Parking Extended*.

Figura 6.18: Taxa de Entraga: mOVERS, *Parking* e *Parking Extended*, LRBF e LRBFAdvanced.

Hora	LRBF	LRBFAdvanced
20	12.5	30
21	52.5	65
22	68.33	75.83
23	70.83	80

Tabela 6.9: Taxa de entrega para o período de *Parking*, LRBF e LRBFAdvanced.

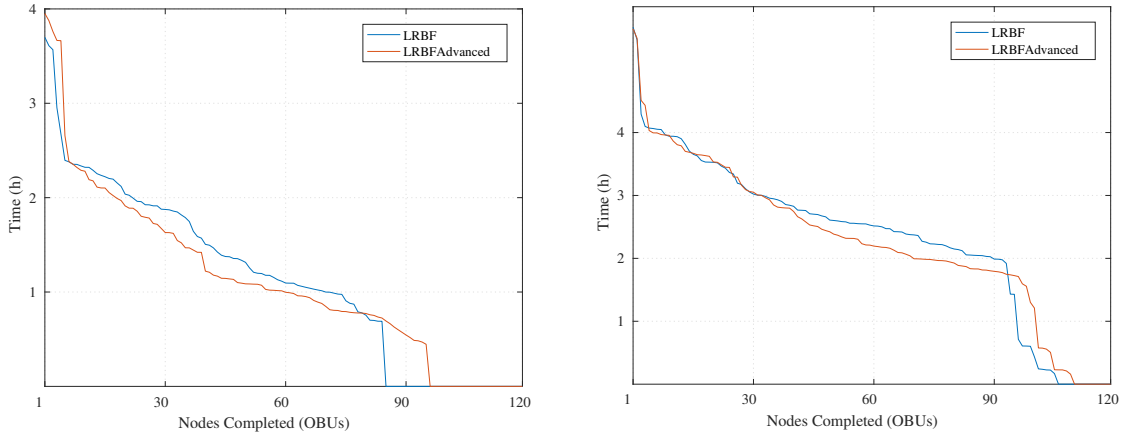
***End-To-End* delay**

A Figura 6.19a confirma os resultados apresentados na métrica anterior, para o período

Hora	LRBF	LRBFAdvanced
18	8.33	7.5
19	13.33	33.33
20	60.83	64.17
21	81.67	87.5
22	86.67	90
23	88.33	91.67

Tabela 6.10: Taxa de entrega para o período de *Parking Extended, LRBF e LRBFAdvanced.*

normal de *Parking*. Por sua vez, a Figura 6.19b confirma os resultados apresentados na métrica anterior, para o período normal de alargado de *Parking*. Como resultado, a estratégia LRBFAdvanced é mais eficiente, uma vez que mais OBUs conseguem completar o ficheiro, devido às várias otimizações efetuadas para diminuir o *overhead* de dados.



(a) *Parking.* (b) *Parking Extended.*

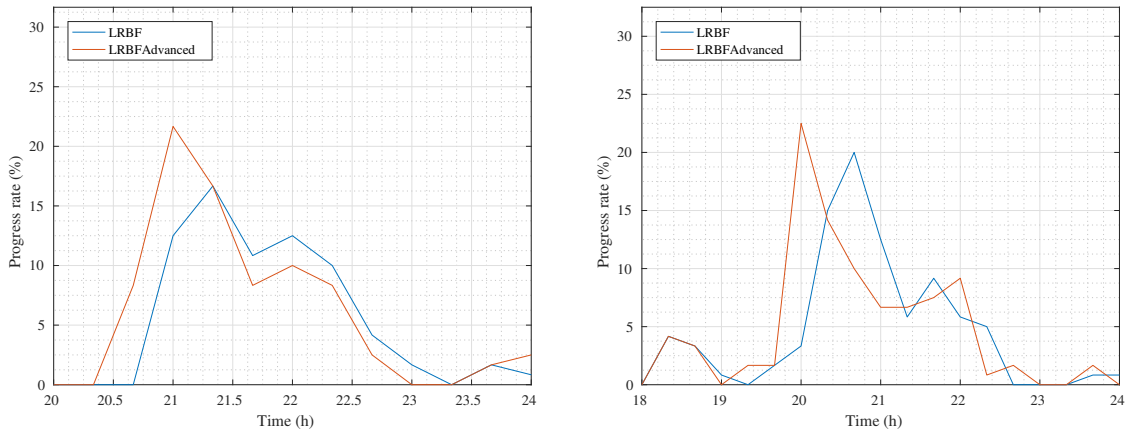
Figura 6.19: *End-To-End* Delay: mOVERS, *Parking* e *Parking Extended*, LRBF e LRBFAdvanced.

Progress Rate

As Figuras 6.20a e 6.20b apresentam os resultados no que diz respeito à velocidade com que o ficheiro está a ser disseminado na rede, para o período normal de *Parking* e para o período alargado de *Parking*. Em ambos os períodos é notório que a estratégia LRBFAdvanced entrega o conteúdo mais rapidamente para os nós da rede, com maior velocidade do que a estratégia LRBF, estando em conformidade com os resultados apresentados nas métricas anteriores.

Número de Pacotes de Dados Transmitidos

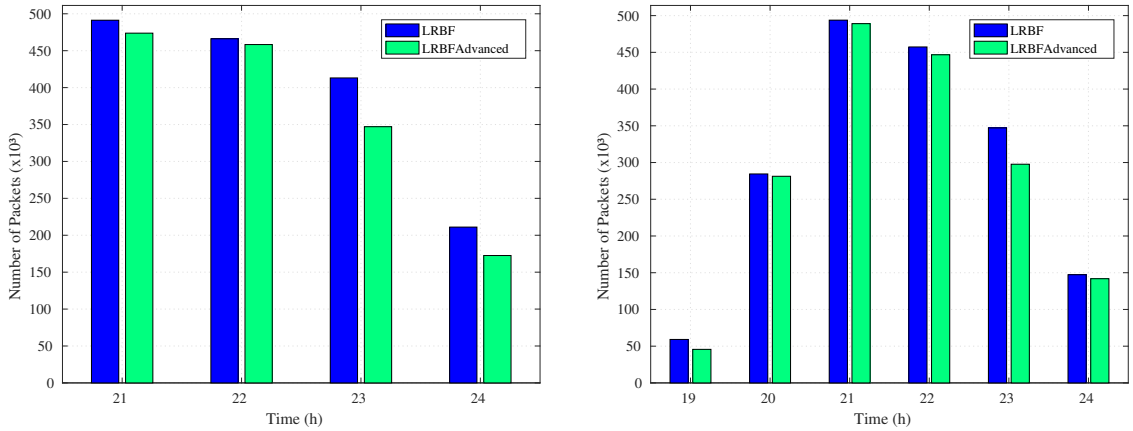
Relativamente à quantidade de pacotes de dados transmitidos, esta é bem maior do que nos outros períodos (*Non Rush* e *Rush Hour*), uma vez que uma grande quantidade de nós está estacionada na mesma área geográfica. Como tal, existe uma maior quantidade de contactos entre os nós da rede. A estratégia LRBFAdvanced apenas conseguiu diminuir cerca de 10% da quantidade enviada pela estratégia LRBF. Este facto é verificado em ambos os períodos de



(a) *Parking*. (b) *Parking Extended*.

Figura 6.20: Progress Rate: mOVERS, *Parking* e *Parking Extended*, LRBF e LRBFAdvanced.

Parking. As Figuras 6.21a e 6.21b ilustram a quantidade de pacotes de dados transmitidos, para o período normal e período alargado.



(a) *Parking*. (b) *Parking Extended*.

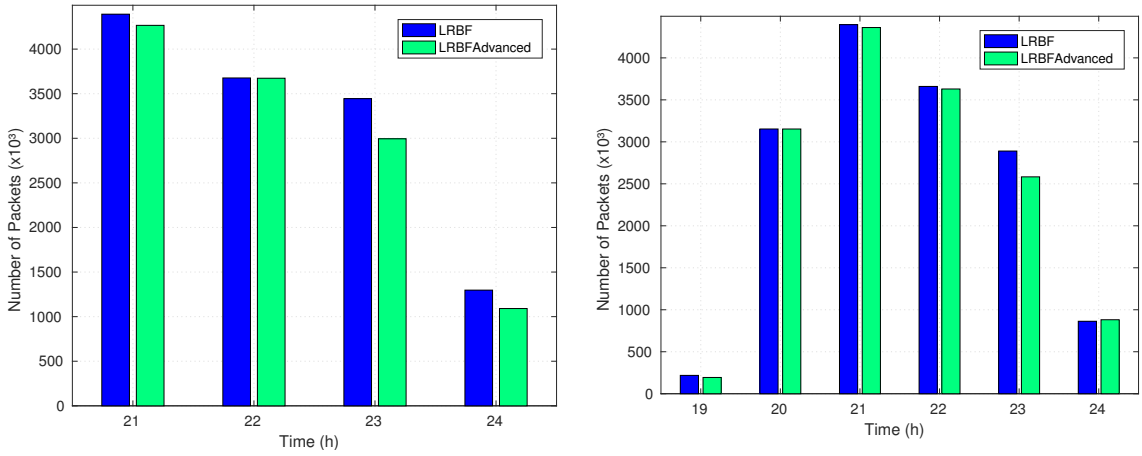
Figura 6.21: Número de Pacotes de Dados Transmitidos: mOVERS, *Parking* e *Parking Extended*, LRBF e LRBFAdvanced.

Pacotes na rede

A Figura 6.22a apresenta a quantidade de pacotes de dados "ouvidos"na rede para o período normal de *Parking*. É esperado que a quantidade seja inferior na estratégia LRBFAdvanced do que na LRBF, uma vez que conseguiu diminuir ligeiramente a quantidade de pacotes de dados enviados, estando assim em concordância com a métrica anterior. O mesmo se verifica para o período de *Parking Extended*.

Número de Pacotes de Controlo Transmitidos

As figuras 6.23a e 6.23b apresentam o número de pacotes de controlo transmitidos, ao longo a experiência, para ambos os períodos de *Parking*. Os resultados mantêm-se bastante

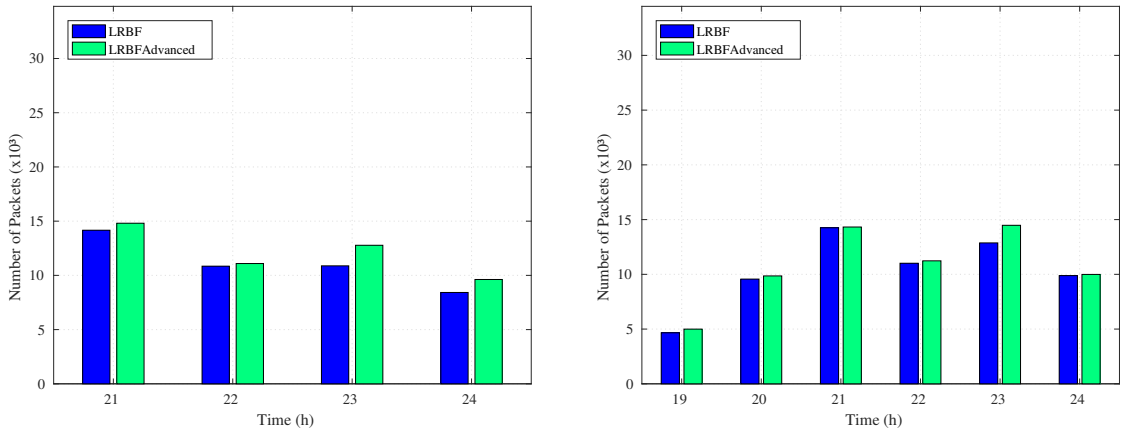


(a) *Parking*. (b) *Parking Extended*.

Figura 6.22: Pacotes na rede: mOVERS, *Parking* e *Parking Extended*, LRBF e LRBFAdvanced.

---

próximos uns dos outros, para ambas as estratégias, como seria de esperar, pois as alterações efetuadas quer ao nível da diminuição o *overhead* de dados quer de controlo não alteram o número de pacotes de controlo enviados. A abordagem *BitArray* diminui o *overhead* de controlo diminuindo o tamanho dos pacotes de controlo.

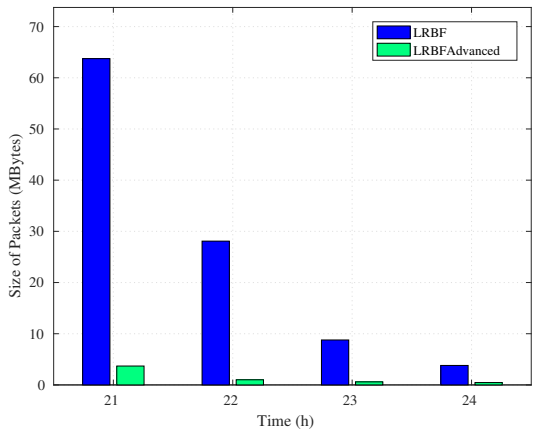


(a) *Parking*. (b) *Parking Extended*.

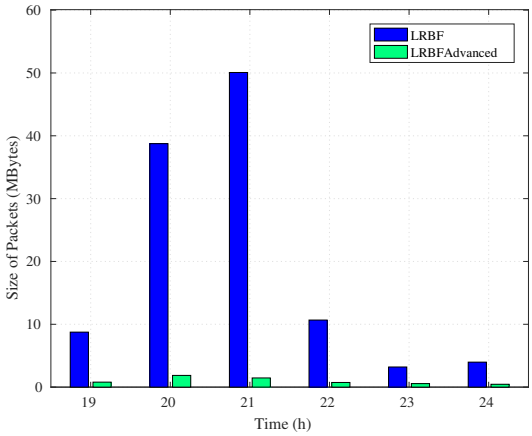
Figura 6.23: Número de Pacotes de Controlo Transmitidos: mOVERS, *Parking* e *Parking Extended*, LRBF e LRBFAdvanced.

Tamanho Total dos Pacotes de Controlo

O tamanho dos pacotes de controlo foi reduzido para 90% do tamanho usado na estratégia LRBF, como mostram os resultados da Figura 6.24a, para o período de *Parking*, e da Figura 6.24b para o período de *Parking Extended*. Mais uma vez, constitui uma prova que a abordagem *BitArray* é bastante eficiente no controlo do congestionamento provocado pelos pacotes de controlo.



(a) *Parking*.



(b) *Parking Extended*.

Figura 6.24: Tamanho Total dos Pacotes de Controlo: mOVERS, *Parking* e *Parking Extended*, LRBF e LRBFAdvanced.

As Tabelas 6.11 e 6.12 apresentam os valores exatos do tamanho dos pacotes de controlo para cada período de *Parking*, para a estratégia LRBF e LRBFAdvanced, respetivamente.

Hora	**LRBF**	**LRBFAdvanced**
21	63.7373	3.6901
22	28.0917	1.0037
23	8.7825	0.5961
24	3.8046	0.4591

Tabela 6.11: Tamanho dos pacotes de controlo, para o período normal de *Parking*.

Hora	**LRBF**	**LRBFAdvanced**
19	8.7529	0.7934
20	38.74	1.8637
21	50.0756	1.4615
22	10.6748	0.7399
23	3.2048	0.5576
24	3.9775	0.4539

Tabela 6.12: Tamanho dos pacotes de controlo, para o período de *Parking Extended*.

6.6 Resultados obtidos no Laboratório

Nesta secção são apresentados os resultados das experiências reais em laboratório em que as OBUs e RSUs são implementadas em equipamentos reais, as placas *NetRiders*.

Antes de o código fonte do mOVERS ser executado nas placas *NetRider*, o canal IEEE

802.11p teve de ser configurado, de forma a tornar possível a comunicação entre os nós da rede. As características da configuração são apresentadas na Tabela 6.13.

Número do Canal	180
Nome do Serviço	80-10
Modo	Continuo
Potência de Transmissão	23 dBm
Bit Rate Máximo	27 Mbps

Tabela 6.13: Especificações de configuração do canal IEEE 802.11p nas Placas *NetRider*.

O ficheiro de configuração que serve como argumento de entrada ao programa foi configurado, especificando os parâmetros mais relevantes, como o número da porta do *Socket* (porta 4556), a capacidade de armazenamento de cada nó (15MB), assim como a interface usada (*wlan1*, correspondente à interface WAVE).

No que diz respeito à configuração das macros relativas à estratégia LRBFAdvanced, estas foram configuradas da seguinte forma: o envio de pacotes de controlo é feito periodicamente a cada 5 segundos; a informação das estruturas internas é válida por um período de 30 segundos, sendo atualizadas a cada 30 segundos (ELEMENT_VALID_TIME = 30). A escrita das informações no ficheiro de log é feita a cada 10 segundos.

Contrariamente às experiências no emulador, na experiência no laboratório não há a limitação de 1Mbps imposta pela largura de banda. No entanto, o envio de pacotes permanece oportunístico, uma vez que um nó só envia pacotes se tiver pelo menos um nó vizinho. Contudo, o módulo *Routing* impõe que um nó adormeça durante um período específico de tempo entre o envio de pacotes. Este período é definido como 100ms, onde um nó poderá enviar um máximo de 10 pacotes por segundo, considerando que cada pacote tem tamanho 32KB.

Para garantir a sincronização entre os nós foi usado o NTP. Este protocolo funciona como um modelo cliente-servidor que é executado em *background*, na RSU como servidor, e nas OBUs como cliente, definindo o servidor com o IP da interface IEEE 802.11p da RSU.

O ficheiro usado para este cenário foi um ficheiro de tamanho 10MB, dividido num total de 320 pacotes, de 32KB em cada pacote.

Os resultados são obtidos de um conjunto de 6 repetições para o cenário descrito em 6.4.1, considerando um intervalo de confiança de 95%.

Taxa de Entrega

A Figura 6.25 apresenta a percentagem de nós que receberam o ficheiro ao longo da experiência para as duas estratégias testadas (LRBF e LRBFAdvanced). Pode-se observar que todos os nós receberam o ficheiro completo durante a experiência, em ambas as estratégias. A taxa de entrega é praticamente a mesma em ambas, levando, no total, 185 segundos para que todas as OBUs completassem o ficheiro. Estes resultados vêem reforçar que as otimizações efetuadas quer ao nível dos pacotes de controlo, quer ao nível dos pacotes de dados, mantêm a taxa elevada de entrega do ficheiro em cenários reais.

A disseminação começa logo no início da experiência, uma vez que, tanto a OBU móvel 645 como a OBU 678 se encontram na área de cobertura da RSU. Nos primeiros 100 segundos, a Figura 6.25 mostra uma grande evolução na taxa de entrega. Este comportamento deve-se ao facto de a OBU 678 já ter completado o ficheiro; a OBU 645 contém metade do ficheiro, e já disseminou essa metade para as restantes OBU fixas 632 e 656. Aos 137 segundos, a OBU

móvel acaba de completar o ficheiro. As restantes OBUs acabam de completar o ficheiro na segunda passagem da OBU móvel, mas agora com o ficheiro completo. Este período é representado na Figura 6.25 pela evolução da taxa de entrega entre os 150 e os 185 segundos.

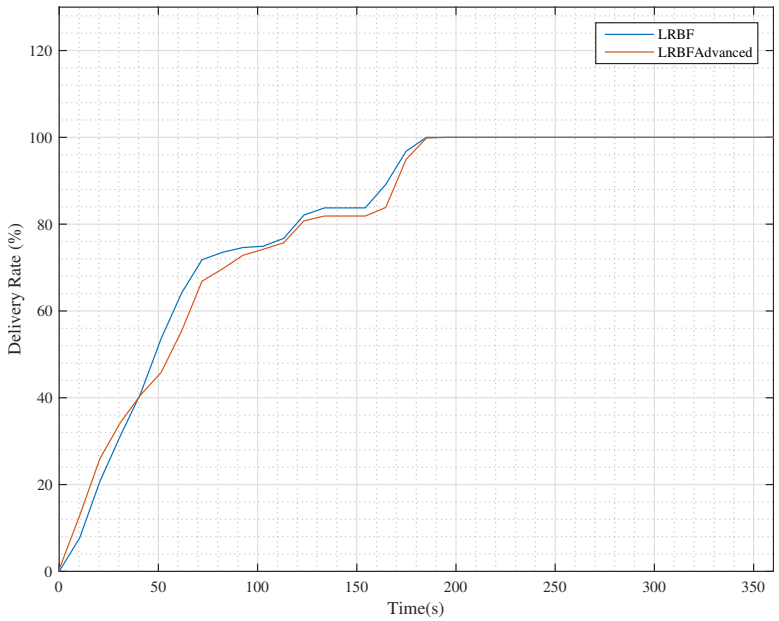


Figura 6.25: Taxa de Entrega: Laboratório, LRBF e LRBFAdvanced.

As Figuras 6.26, 6.27, 6.28 e 6.29 confirmam a análise efetuada anteriormente, uma vez que descrevem a taxa de entrega de cada OBU, no que diz respeito a cada estratégia durante toda a experiência.

O primeiro nó a receber a totalidade do ficheiro é a OBU fixa 678, uma vez que se encontra na área de cobertura da RSU. Este comportamento é esperado para ambas as estratégias.

O segundo nó a completar o ficheiro é a OBU móvel 645, devido aos períodos de contacto direto com a RSU. Este comportamento é verificado em ambas as estratégias, não havendo diferenças significativas entre elas. Por último, as OBUs fixas 632 e 656 acabam por completar o ficheiro ao fim de pouco tempo depois, para ambas as estratégias.

End-To-End delay

A Figura 6.30 apresenta a média de tempo que cada nó leva para receber o ficheiro completo. Assim sendo, é possível concluir que a OBU 678 completa o ficheiro mais rápido na estratégia LRBFAdvanced do que na LRBF. A média de tempo para a segunda OBU completar o ficheiro (645) é igual em ambas as estratégias. As duas OBUs fixas fora da área de cobertura da RSU são ligeiramente mais rápidas a receber o ficheiro na estratégia LRBF.

Número de Pacotes de Dados Transmitidos

A Figura 6.31 ilustra a quantidade de pacotes de dados transmitidos por cada OBU. Esta métrica é uma das métricas importantes para avaliar a nova estratégia implementada, a LRBFAdvanced. Ao contrário das experiências no emulador, o controlo do *overhead* a nível destes pacotes é mais notório na prática. Na estratégia LRBF verifica-se que existe um maior *overhead* na rede do que na estratégia LRBFAdvanced. Esta métrica constitui uma evidência

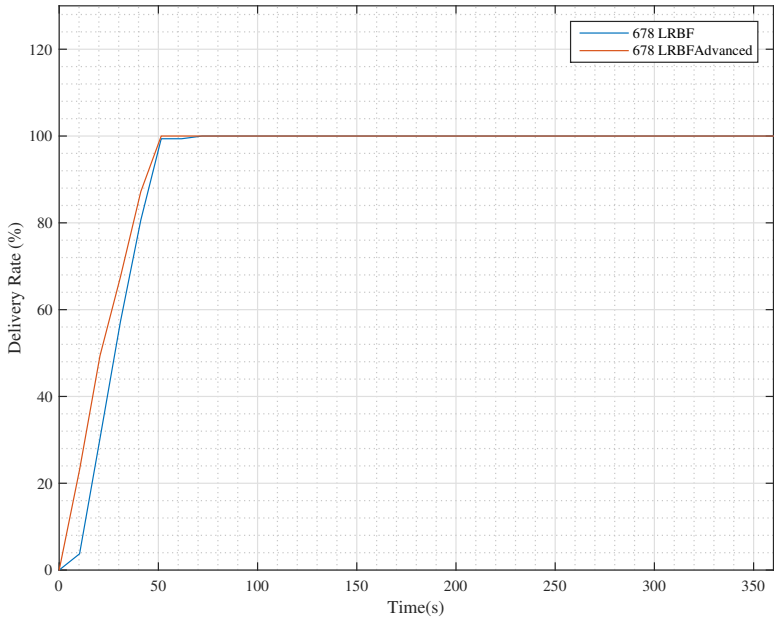


Figura 6.26: Percentagem de ficheiro distribuído pela OBU 678: Laboratório, LRBF e LRBFAdvanced.

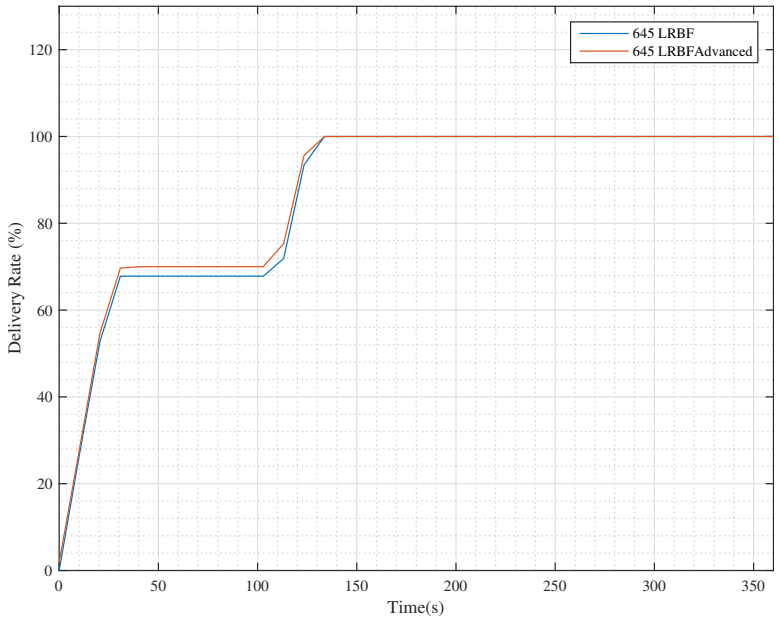


Figura 6.27: Percentagem de ficheiro distribuído pela OBU 645: Laboratório, LRBF e LRBFAdvanced.

---

de que o objetivo de diminuir o problema de *broadcast storm* foi conseguido em comparação com a estratégia padrão. Como seria de esperar, a OBU 678 encontra-se na área de cobertura da RSU e, por isso, não transmite pacotes de dados. A transmissão de pacotes de dados nos nós 645, 656 e 650, permanece dentro dos mesmos valores para a estratégia LRBFAdvanced, sendo que a OBU 632 é a que transmite ligeiramente mais dados. Também para a estratégia

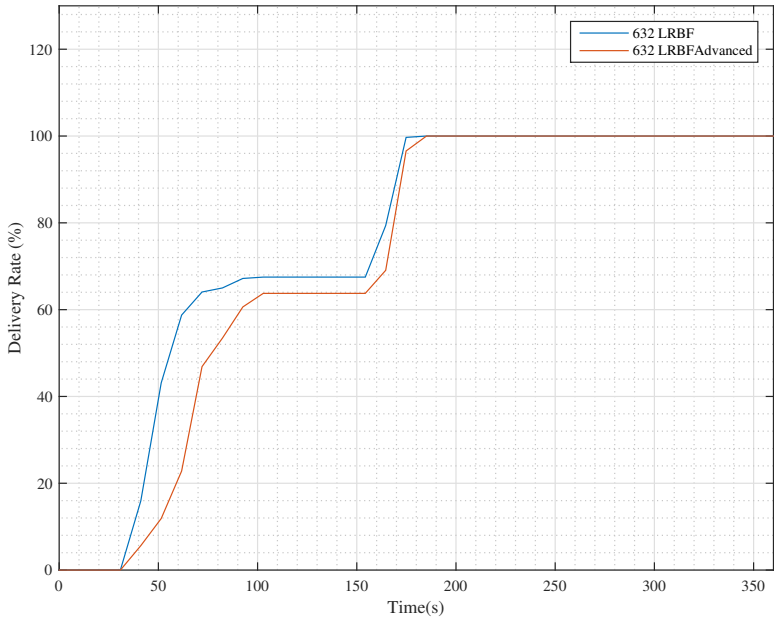


Figura 6.28: Percentagem de ficheiro distribuído pela OBU 632: Laboratório, LRBF e LRBFAdvanced.

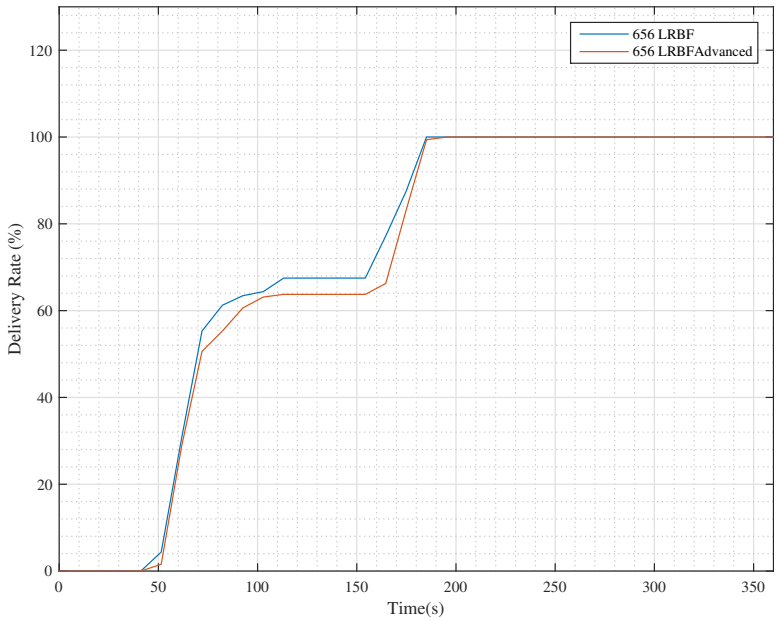


Figura 6.29: Percentagem de ficheiro distribuído pela OBU 656: Laboratório, LRBF e LRBFAdvanced.

LRBF, a OBU que transmite mais dados é a 632, mas com um maior número de vezes, seguido da OBU 656.

No total a estratégia LRBFAdvanced conseguiu diminuir o tráfego de dados em 56% quando comparada com a estratégia LRBF.

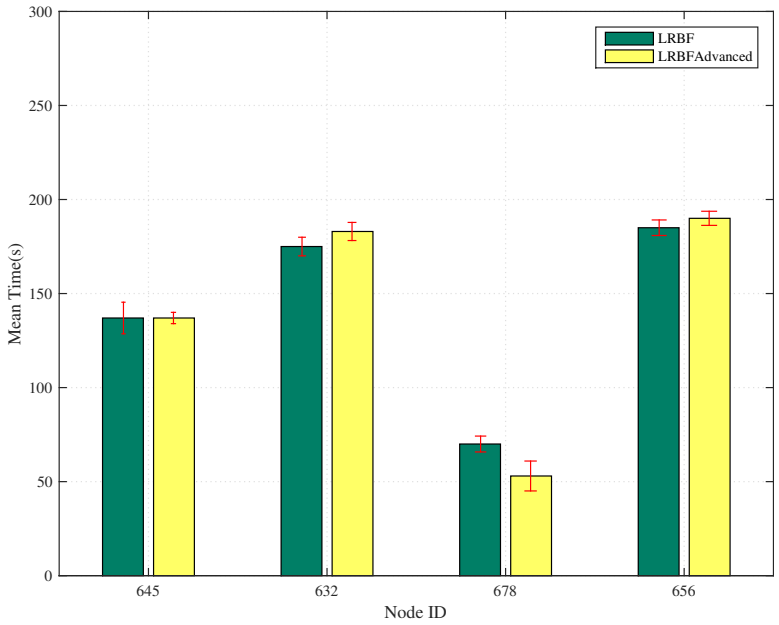


Figura 6.30: End-to-End delay: Laboratório, LRBF e LRBFAdvanced.

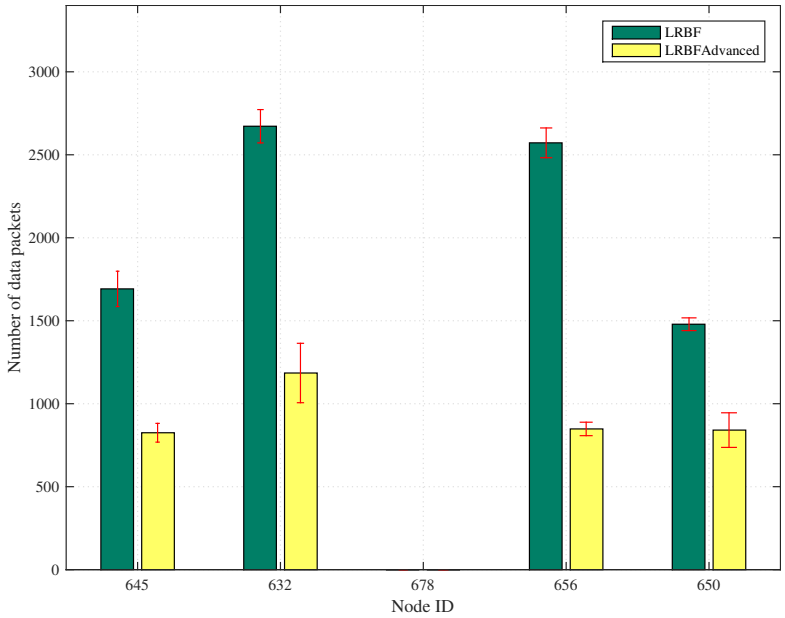


Figura 6.31: Número de Pacotes de Dados Transmitidos: Laboratório, LRBF e LRBFAdvanced.

Pacotes na Rede

O número de pacotes "ouvidos"na rede por cada nó é representado na Figura 6.32. Dado que o número de pacotes de dados enviados na estratégia LRBFAdvanced foi menor, é esperado que o número de pacotes "ouvidos"na rede seja menor, como mostra a Figura. Pelo contrário, na estratégia LRBF, como foi enviada uma maior quantidade de pacotes de dados,

é normal que os pacotes "ouvidos"na rede seja maior. A OBU 678 foi a que "ouviu"uma menor quantidade, dado que apenas "ouve"pacotes da RSU e da OBU 645. A quantidade de pacotes "ouvidos"é uniforme entre as restantes OBUs, uma vez que "ouvem"pacotes umas das outras. A RSU apenas envia pacotes até que a OBU móvel e a OBU 678 tenham completado o ficheiro.

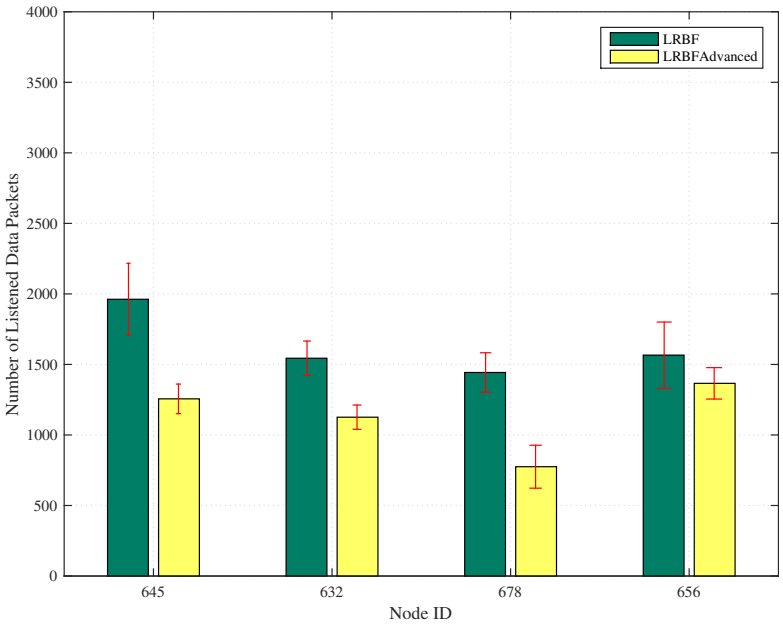


Figura 6.32: PAcotes na Rede: Laboratório, LRBF e LRBFAdvanced.

Número de Pacotes de Controlo Transmitidos

O número de pacotes de controlo enviados por cada nó é representado na Figura 6.33. Para ambas as estratégias, o número de pacotes de controlo enviados é similar, embora com algumas variações, mas não significativas. Este comportamento é esperado, uma vez que o envio destes pacotes é periódico e é despoletado pela receção de um pacote de controlo com informação sobre o novo conteúdo a ser disseminado na rede. A OBU 656 é a que envia menos pacotes de controlo, pois é a última a receber a informação sobre um novo ficheiro.

Tamanho Total dos Pacotes de Controlo

A Figura 6.34 ilustra o tamanho total dos pacotes de controlo enviados por cada OBU. Esta é outra das métricas mais relevantes para avaliar o controlo do *overhead* ao nível dos pacotes de controlo. Da mesma forma que nas experiências no emulador, é notória a diminuição do tamanho dos pacotes de controlo em mais de 90% em comparação com a estratégia *LRBF*. Esta é mais uma evidência de que a abordagem funciona na prática para qualquer que seja o tamanho do ficheiro a ser disseminado. Para a RSU de ambas as estratégias é esperado que o tamanho seja igual para os dois casos, pelas mesmas razões explicadas em 6.5.1.1. Como a RSU tem o ficheiro completo, apenas dissemina essa informação, utilizando apenas 16 bytes.

As Tabelas 6.14 e 6.15 combinam a informação do tamanho total dos pacotes de controlo enviados com o tamanho total dos pacotes de dados multiplicado pelo número de vezes que foram transmitidos, discriminando o peso de cada um, para ambas as estratégias testadas em

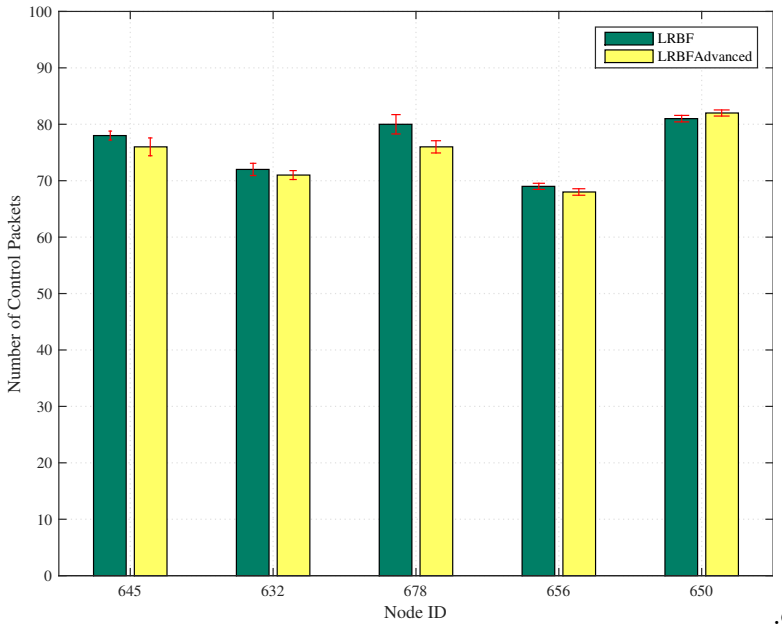


Figura 6.33: Número de Pacotes de Controlo Transmitidos: Laboratório, LRBF e LRBFAdvanced.

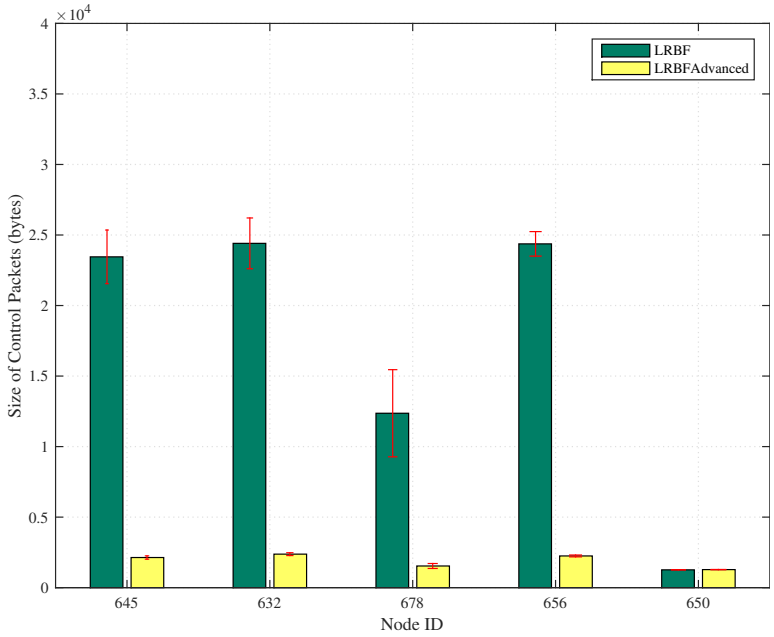


Figura 6.34: Tamanho Total dos Pacotes de Controlo: Laboratório, LRBF e LRBFAdvanced.

laboratório.

Como tal, é possível concluir que em ambas as estratégias, os pacotes de controlo adicionam um pequeno *overhead* à rede veicular, uma vez que a maioria dos pacotes em disseminação são pacotes de dados. Comparando ambas as Tabelas conclui-se que a estratégia LRBFAdvanced conseguiu controlar o problema de *broadcast storm*, tanto ao nível do tráfego

Nó	**Tamanho Total Pacotes Transmitidos** $(Controlo + Dados)[KB]$	**Pacotes de Controlo** $[KB](\%)$	**Pacotes de Dados** $[KB](\%)$
645	63766.9	22.9 (0.036%)	63744 (99.94%)
632	85527.8	23.8 (0.028%)	85504 (99.97%)
678	12.1	12.1 (100%)	0 (0.0%)
656	82327.8	23.8 (0.029%)	823204 (99.97%)

Tabela 6.14: Avaliação do impacto dos pacotes de controlo no processo de disseminação na estratégia LRBF.

Nó	**Tamanho Total Pacotes Transmitidos** $(Controlo + Dados)[KB]$	**Pacotes de Controlo** $[KB](\%)$	**Pacotes de Dados** $[KB](\%)$
645	26402.1	2.1 (0.0078%)	26400 (99.99%)
632	37922.3	2.3 (0.006%)	37920 (99.99%)
678	1.5	1.5 (100%)	0 (0.0%)
656	27138.2	2.2 (0.008%)	27136 (99.99%)

Tabela 6.15: Avaliação do impacto dos pacotes de controlo no processo de disseminação na estratégia LRBFAdvanced.

de controlo como do tráfego de dados.

6.7 Resultados obtidos nos Moliceiros da ria de Aveiro

Nesta secção são apresentados os resultados da experiência real realizada nos moliceiros da ria de Aveiro. As configurações foram as mesmas que foram feitas para o cenário de laboratório. No entanto, foi necessário fazer algumas alterações, uma vez que a duração da experiência foi muito maior. As alterações efetuadas foram o aumento do tamanho do ficheiro para 40MB, dividido em 1280 pacotes de 32KB cada; e a disseminação de vários ficheiros de 40MB na rede, através do *reset* das estruturas de dados internas a cada 45 minutos.

A experiência começa com a disseminação do primeiro ficheiro na rede. Após 45 minutos é iniciada a disseminação do segundo ficheiro e assim sucessivamente. Todos os dados referentes a cada ficheiro são guardados, imediatamente antes do *reset* das estruturas, para posterior análise.

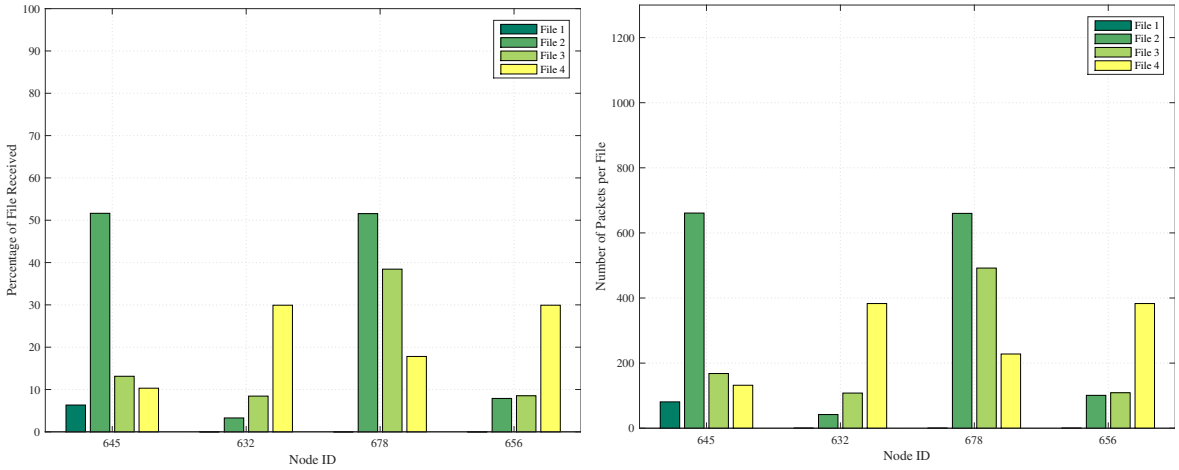
O período de tempo de *reset* das estruturas de dados internas escolhido foi 45 minutos, uma vez que a duração de cada passeio é estimada em cerca de 40 minutos (informação obtida junto de cada companhia). A duração da experiência foi 3h, dado que é a duração máxima das baterias usadas. O número total de ficheiros em disseminação na rede foi de 4 ficheiros, e apenas foi feito um ensaio na data de 13 de Julho de 2018, cujo principal objetivo foi a recolha das informações relativas aos tempos de contacto dos moliceiros.

Percentagem e número total de pacotes por cada ficheiro recebido

A primeira métrica mede a percentagem obtida de cada ficheiro por cada OBU, cujos resultados são apresentados na Figura 6.35a. A segunda métrica diz respeito ao número

de pacotes correspondente a cada percentagem, cujos resultados são apresentados na Figura 6.35b).

Conclui-se que nenhuma OBU conseguiu receber completamente um dos 4 ficheiros ao longo da experiência. O melhor resultado foi conseguido para o Ficheiro 2, para as OBUs 645 e 678, que conseguiram completar um pouco mais de metade do ficheiro (661 e 660 pacotes, respetivamente). O pior resultado foi para o Ficheiro 1, em que apenas a OBU 645 recebeu pacotes deste ficheiro. Mesmo assim, a quantidade recebida foi muito baixa (81 pacotes - 6%). Como resultado, a taxa de entrega de cada ficheiro foi muito baixa.



(a) Percentagem de ficheiros recebidos.

(b) Número total de pacotes por ficheiro.

Figura 6.35: Percentagem e número total de pacotes de cada ficheiro: Moliceiros, LRBFAdvanced.

Para analisar em profundidade as razões que podem ter levado à taxa reduzida de entrega foi necessário estudar os tempos de contacto obtidos entre cada nó da rede e os pacotes recebidos durante esse tempo de contacto.

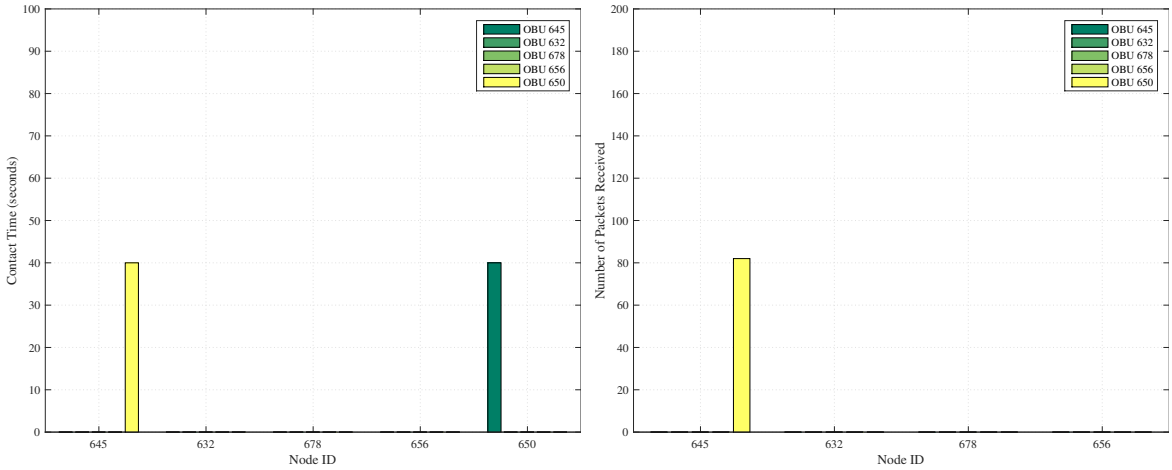
Tempo de contacto e quantidade de pacotes recebidos

A primeira métrica mede o tempo de contacto cumulativo em segundos entre cada nó da rede, e a segunda métrica mede a quantidade total de pacotes em cada conjunto de contactos, para um determinado ficheiro, ou seja, os pacotes recebidos durante cada tempo de contacto acumulativo.

As Figuras 6.36a e 6.36b apresentam os resultados obtidos para o ficheiro 1. As Figuras 6.37a, 6.37b, 6.38a, 6.38b, 6.39a, e 6.39b apresentam os resultados obtidos para os ficheiros 2, 3 e 4, respetivamente.

Analisando as Figuras 6.36a e 6.36b, apenas a OBU 645 teve contacto com a RSU, o que justifica a razão pela qual mais nenhuma OBU ter recebido pacotes do Ficheiro 1. No entanto, o tempo de contacto entre a OBU 645 e a RSU foi relativamente baixo (40 segundos), o que justifica a impossibilidade desta OBU ter recebido a totalidade do ficheiro. A inexistência de contacto nas OBUs 656, 678 e 632 e o baixo tempo de contacto da OBU 645 justificam a taxa de entrega quase nula relativamente ao primeiro ficheiro.

Relativamente ao Ficheiro 2, a OBU 645 obteve 661 pacotes no total, dado que a grande maioria foi obtida durante o período de contacto (130 segundos) com a RSU (457 pacotes),

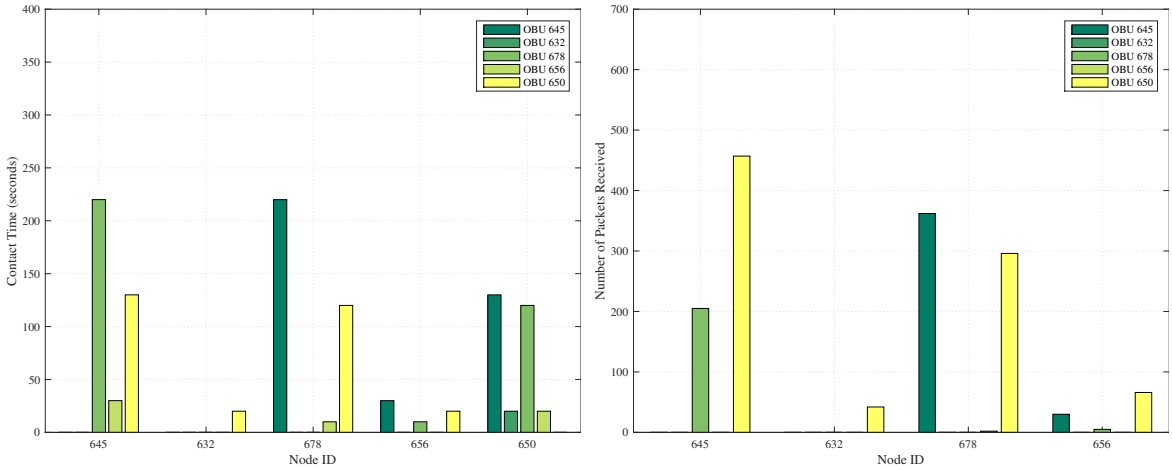


(a) Tempo de contacto. (b) Quantidade de pacotes.

Figura 6.36: Tempo de contacto e quantidade de pacotes em cada conjunto de contactos para o Ficheiro 1: Moliceiros, LRBFAdvanced.

---

sendo a restante quantidade obtida durante o período de contacto (220 segundos) com a OBU 678 (204 pacotes), não havendo repetições de pacotes. Apesar de existir um tempo de contacto de 30 segundos com a OBU 656, a OBU 645 não obteve nenhum pacote desta, o que significa que apenas enviou pacotes. A OBU 678 obteve um total de 660 pacotes, sendo que a maioria foi obtida durante o tempo de contacto com a OBU 645 (362 pacotes); a restante quantidade foi obtida durante o tempo de contacto (120 segundos) com a RSU (296 pacotes) e com a OBU 656 (10 segundo - 102 pacotes). Por sua vez, a OBU 656 obteve um total de 101 pacotes, 30 pacotes durante o tempo de contacto com a OBU 645, 5 pacotes durante o tempo de contacto com a OBU 678 e 66 pacotes da RSU. A OBU 632 apenas teve um contacto de 20 segundos com a RSU onde obteve 42 pacotes, como mostram as Figuras 6.37a e 6.37b.



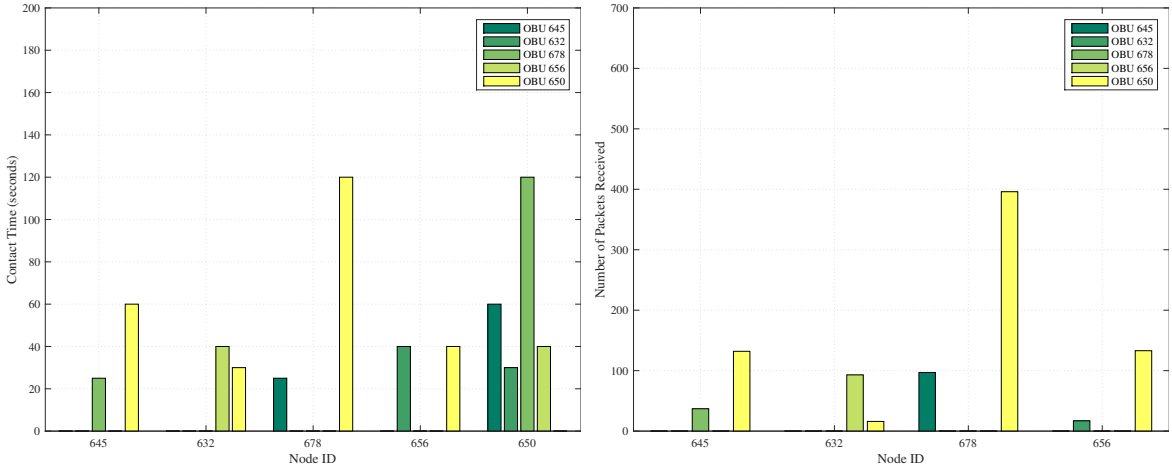
(a) Tempo de contacto. (b) Quantidade de pacotes.

Figura 6.37: Tempo de contacto e quantidade de pacotes em cada conjunto de contactos para o Ficheiro 2: Moliceiros, LRBFAdvanced.

Para além do pouco tempo de contacto entre os nós da rede, principalmente com a RSU, uma vez que a disseminação dos ficheiros parte desta infraestrutura, outra das razões para explicar o porquê de nenhuma OBU ter completado o ficheiro é o método adotado para simular a disseminação de vários ficheiros na rede: o *reset* das estruturas de dados internas. É esperado que cada OBU complete cada ficheiro num tempo máximo de 45 minutos, o que pode não ser suficiente para um ficheiro de 40MB, devido à aleatoriedade do cenário.

A falta de contacto entre alguns dos nós da rede poderá também ter sido devido aos moliceiros terem trajetórias com direções opostas.

Como tal, os resultados apresentados nas Figuras 6.38a, 6.38b, 6.39a e 6.39b apresentam as mesmas caraterísticas aos apresentados nas figuras anteriores. Quanto maior é o tempo de contacto, maior é a quantidade de pacotes recebidos nos conjuntos, embora não sendo o suficiente para que os nós completem o ficheiro em disseminação. Se o tempo de contacto é baixo, é esperado que as OBUs não recebam a totalidade do ficheiro. Quanto menor for o tempo de contacto entre as OBUs e a RSU, maior é a probabilidade de não receberem o ficheiro completo, mesmo que o tempo de contacto entre elas seja superior ao da RSU, como seria de esperar. Por outro lado, os valores de RSSI variam ao longo do tempo de contacto entre os nós da rede, podendo este contacto ser traduzido em conetividade intermitente.



(a) Tempo de contacto.

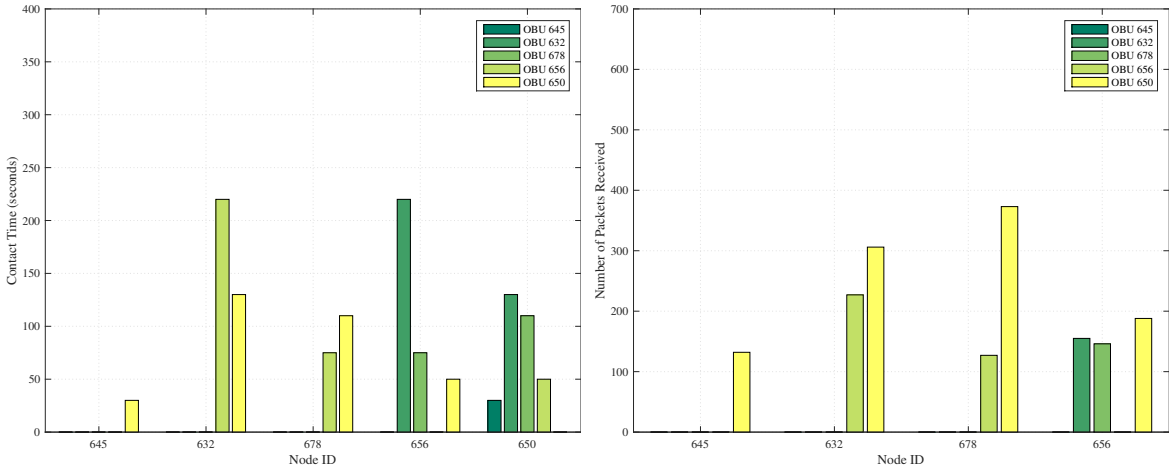
(b) Quantidade de pacotes.

Figura 6.38: Tempo de contacto e quantidade de pacotes em cada conjunto de contactos para o Ficheiro 3: Moliceiros, LRBFAdvanced.

Em suma, apenas a realização de um ensaio não é suficiente para tirar todas as conclusões necessárias, mais ensaios devem ser realizados no futuro. O tempo de *reset* das estruturas de dados deve ser repensado, por forma a encontrar um tempo mais adequado, que permita a receção completa do ficheiro. Outra sugestão seria aumentar o número de RSUs para 2, com vista a colmatar o baixo tempo de contacto entre os nós da rede e a infraestrutura.

6.8 Considerações Finais

Neste capítulo foram avaliadas as estratégias propostas no capítulo anterior. Inicialmente, apenas as estratégias com vista à diminuição da sobrecarga introduzida pelos pacotes de controlo foram avaliadas, para os períodos de *Rush e Non Rush Hour*, através da experiência



(a) Tempo de contacto.

(b) Quantidade de pacotes.

Figura 6.39: Tempo de contacto e quantidade de pacotes em cada conjunto de contactos para o Ficheiro 4: Moliceiros, LRBFAdvanced.

usando o mOVERS. Os resultados obtidos confirmaram que a estratégia com melhor desempenho foi, de facto, a estratégia *BitArray*. De seguida, foi avaliada a nova estratégia proposta, LRBFAdvanced, consistindo na abordagem *BitArray* (abordagem com melhor desempenho ao nível dos pacotes de controlo) e nas várias otimizações ao nível dos pacotes de dados efetuadas na estratégia LRBF, para os períodos de *Rush e Non Rush Hour*, e também para um novo cenário, o período de *Parking*. As alterações efetuadas ao nível da largura de banda permitiram a execução de testes para este período, caraterizado por um grande número de contactos entre os nós da rede, levando a um aumento significativo do congestionamento na rede. Os resultados obtidos mostraram que a estratégia LRBFAdvanced diminui o tráfego de dados e controlo para os períodos de *Rush, Non Rush Hour e Parking*, sendo que a diminuição do tráfego de dados é mais evidente para os dois primeiros. Para o cenário de *Parking*, a diminuição do número de pacotes redundantes na rede é traduzida também pelo aumento da taxa de entrega, em relação à estratégia LRBF. Em relação ao tráfego de controlo, este foi reduzido para cerca de 90% do seu valor inicial.

Por fim, a estratégia LRBFAdvanced foi avaliada num cenário real juntamente com a estratégia base, LRBF. O objetivo principal deste teste foi confirmar a eficácia da estratégia LRBFAdvanced na diminuição do tráfego de dados e controlo em relação à estratégia base, LRBF, num cenário real. O teste foi realizado em laboratório utilizando as mesmas OBUs utilizadas na rede veicular do Porto. A avaliação confirmou o correto desenvolvimento e implementação da estratégia proposta, bem como o seu principal objetivo, estando, assim, pronta para ser implementada e avaliada na rede veicular da cidade do Porto. A estratégia LRBFAdvanced foi avaliada num cenário real não controlado, nos moliceiros da ria de Aveiro. Foram feitas algumas modificações de modo a implementar a disseminação de vários ficheiros na rede. Os resultados obtidos mostram que o tempo de receção de um ficheiro de 40MB é superior a 45 minutos, devido ao baixo tempo de contacto entre os nós da rede, principalmente com a RSU.

Capítulo 7

Conclusões e Trabalho Futuro

7.1 Conclusões

As estratégias de distribuição de conteúdos como a estratégia LRBF, para as redes DTNs, atingem uma taxa de entrega bastante elevada, e um baixo atraso, através da seleção da informação correta a enviar. No entanto, esta seleção depende do conhecimento completo da informação presente nos nós vizinhos. Para isso, é introduzida nos pacotes de controlo uma lista completa de todos os pacotes (*hashes*) que o nó já tem, levando ao aumento significativo do seu tamanho.

Para mapear as *hashes* de cada pacote, no *payload* do pacote de controlo, são necessários 4 bytes, o que é extremamente penoso para ficheiros com tamanhos elevados.

Esta dissertação propõe quatro estratégias para reduzir o tamanho dos pacotes de controlo de forma a conseguir controlar a sobrecarga introduzida por estes pacotes na rede, sem diminuir o desempenho da estratégia a que se aplicam.

As estratégias propostas podem ser usadas em qualquer estratégia de disseminação de conteúdos, sempre que haja a troca periódica de pacotes de controlo, com a informação completa sobre os recursos dos vizinhos. Para demonstrar a eficácia das soluções propostas, estas foram aplicadas na estratégia LRBF, uma vez que foi a estratégia que apresentou uma maior taxa de entrega, mas à custa da introdução de sobrecarga de controlo na rede.

A primeira estratégia proposta, designada de *Bands*, tem em vista o envio de informação na forma de bandas (ou faixas), isto é, para conjuntos sequenciais formado por pacotes em ordem ascendente, apenas considera necessário enviar o primeiro e último elemento do conjunto, diminuindo assim o tamanho do pacote de controlo. A segunda estratégia, *BitArray*, usa um array de *bits* para codificar a informação presente no nó, onde cada *bit* indica se o pacote já foi ou não recebido. Por último, as duas últimas estratégias usam uma estrutura de dados para resumir o conteúdo, denominada *Bloom filter*. A estratégia designada *BFfixedSize* usa um filtro de tamanho fixo, e a estratégia chamada *BFvariableSize* usa um filtro de tamanho variável, pois considera que o parâmetro correspondente ao número máximo de elementos que se pode inserir no filtro (*projectElementCount*) é igual ao número de pacotes que o nó tem para um determinado momento. Na estratégia *BFfixedSize* este parâmetro é fixo, uma vez que é igual ao número total de pacotes que o ficheiro ocupa.

A estratégia que obteve o melhor desempenho foi a estratégia *BitArray*, uma vez que apenas usa 1 bit para codificar a informação de cada pacote.

Por outro lado, as várias otimizações feitas ao nível dos pacotes de dados na estratégia

LRBF, de forma a combater o problema de *broadcast storm*, deram origem ao aparecimento de uma nova estratégia, LRBFAdvanced. As otimizações efetuadas foram: sempre que um nó envia um pacote de dados marca esse pacote como enviado; sempre que um nó verifica que o número total de vizinhos é igual ao tamanho total da lista que indica quantos vizinhos têm um determinado pacote, não envia esse pacote, pois todos eles já têm esse pacote; sempre que um nó "ouve"e/ou recebe um pacote de dados na rede, marca esse pacote como enviado, uma vez que o envio de pacotes é feito em *broadcast*; sempre que um nó recebe um pacote de controlo é feito o *reset* das *flags* dos pacotes enviados que o nó emissor ainda não tem; este *reset* também é feito sempre que o nó deixa de ter contacto com os vizinhos que continham esse pacote.

Esta estratégia engloba também a estratégia de otimização dos pacotes de controlo que obteve melhor desempenho, o *BitArray*. Assim, foi possível concluir que, para os períodos de *Rush* e *Non Rush Hour*, a estratégia LRBFAdvanced diminuiu o número de pacotes de dados enviados, bem como o tamanho dos pacotes de controlo, diminuindo assim a sobrecarga introduzida na rede. Para o período de *Parking*, para além de diminuir significativamente o tamanho dos pacotes de controlo, as várias otimizações feitas ao nível dos pacotes de dados foram evidenciadas, não tanto pela diminuição do número dos pacotes de dados, mas sim pelo aumento da taxa de entrega e do atraso, sendo mais eficaz na entrega de conteúdos.

Após a escolha da melhor estratégia foram realizados testes reais no Instituto de Telecomunicações, através da integração do código fonte do emulador nas placas reais. Estes testes consistiram em avaliar o comportamento das estratégias LRBF e LRBFAdvanced numa rede real. Os resultados foram os esperados: a estratégia LRBFAdvanced reduziu significativamente o tamanho dos pacotes de controlo e o número de pacotes de dados enviados, mantendo a taxa de entrega elevada.

Foi também realizado um ensaio com a estratégia LRBFAdvanced numa rede real não controlada com conteúdos e utilizadores reais, nos moliceiros da ria de Aveiro. Os os resultados obtidos mostram que a estratégia LRBFAdvanced funciona numa rede real não controlada. Neste cenário real nenhuma das OBUs conseguiu completar um dos quatro ficheiros ao longo da experiência. Este facto está relacionado com o método adotado para simular a disseminação de vários ficheiros na rede: o *reset* das estruturas de dados internas a cada 45 minutos. Este tempo pode não ser suficiente para a receção de um ficheiro completo de 40 MB, devido aos tempos reduzidos de contacto entre os nós da rede (ver secção 6.7), principalmente com a RSU, para cada ficheiro, e devido também à aleatoriadade do cenário. Em testes futuros terá de se adequar o tempo de *reset* para permitir a receção completa de um ficheiro de 40MB. Outra sugestão seria aumentar o número de RSUs para 2, com vista a colmatar o tempo reduzido de contacto entre os nós da rede e a infraestrutura.

Vários desafios surgiram durante o desenvolvimento desta Dissertação, a maioria dos quais relacionados com a análise do código do emulador, a fim de compreender adequadamente todos os procedimentos necessários à execução de uma experiência bem como à implementação das várias abordagens em si. Além disso, outro dos desafios sentidos foi a integração do código do emulador nas placas reais devido à falta de documentação existente.

O objetivo desta dissertação foi conseguido, uma vez que, o tamanho dos pacotes de controlo foi significativamente reduzido em 90% do seu tamanho inicial, através da estratégia *BitArray*, assim como, o tráfego de dados onde os testes reais evidenciam uma redução de 56% em relação à estratégia LRBF.

Em suma, é possível concluir que o problema pela qual esta dissertação visa resolver foi endereçado com sucesso em redes reais.

7.2 Trabalho Futuro

A área de distribuição de conteúdos não urgente ainda não foi totalmente explorada e, como tal, existem ainda muitos tópicos que merecem ser estudados e avaliados. Estudos futuros nesta área devem incluir as seguintes melhorias:

- **Avaliar o impacto de ter diferentes tipos de conteúdos a serem distribuídos**. Esta dissertação apenas analisou e avaliou o impacto da disseminação de apenas um ficheiro na rede. No entanto, é necessário o estudo e análise do impacto de vários ficheiros em disseminação na rede com diferentes tamanhos de pacotes de dados.
- **Diferenciação na entrega de conteúdos diferentes**. Ter diferentes conteúdos em disseminação na rede com diferentes prioridades de entrega.
- **Proposta de estratégias que permitam inferir o conteúdo enviado e diminuir o número de pacotes de controlo.** Estas estratégias consideram que é enviada toda a informação relacionada com os pacotes presentes em cada nó, mas a inferência desta informação com novas estratégias pode melhorar significativamente o overhead de controlo.
- **Desempenho do emulador mOVERS**. O emulador usado é executado num servidor que requer a partilha de recursos, podendo sofrer uma possível interrupção abrupta. Dado que o mOVERS requer uma elevada quantidade de tempo para emular um determinado período de tempo, o seu código deve ser avaliado na tentativa de reduzir o tempo necessário para experiência tornando o emulador mais robusto.
- **Evolução para uma rede de distribuição de conteúdos.** Solução híbrida para distribuição de conteúdos na rede veicular que estenda e adapte os conceitos vindos das CDNs à rede veicular, através da integração de réplicas alocadas nos veículos e não só nos servidores estáticos.
- **Realização de mais testes numa rede real de elevada escala com conteúdos e utilizadores reais, como por exemplo os moliceiros da ria de Aveiro.** Esta dissertação apenas apresentou um ensaio realizado nos moliceiros da ria de Aveiro. Mais testes são necessários para tirar mais conclusões.

Bibliografia

[1] Hamssa Hasrouny, Abed Ellatif Samhat, Carole Bassil, and Anis Laouiti. Vanet security challenges and solutions: A survey. *Vehicular Communications*, 7:7 – 20, 2017.

[2] Mohamed Nidhal Mejri, Jalel Ben-Othman, and Mohamed Hamdi. Survey on vanet security challenges and possible cryptographic solutions. *Vehicular Communications*, 1(2):53 – 66, 2014.

[3] R. A. Uzcategui, A. J. De Sucre, and G. Acosta-Marum. Wave: A tutorial. *IEEE Communications Magazine*, 47(5):126–133, May 2009.

[4] A. Cardote. *Plataforma de comunicações veiculares com infraestrutura de suporte*. Ph.D. dissertation, Instituto de Telecomunicac oes, Universidade Aveiro, 2014.

[5] Forrest Warthman. Delay- and Disruption-Tolerant. Networks (DTNs). A Tutorial. Version 3.2. September 14, 2015. [Online] Disponível: http://ipnsig.org/wp-content/uploads/2015/09/DTN_Tutorial_v3.2.pdf, Maio 2018.

[6] Paulo Rogério Pereira, Augusto Casaca, Joel J. P. C. Rodrigues, Vasco N. G. J. Soares, Joan Triay, and Cristina Cervello-Pastor. From delay-tolerant networks to vehicular delay-tolerant networks. *IEEE Communications Surveys and Tutorials*, 14(4):1166–1182, 2012.

[7] Y. Zhao. *B-sub: A practical bloom-filter-based publish-subscribe system for human networks.* International Conference on Distributed Computing Systems, 2010.

[8] K. Na Nakorn, Y. Ji, and K. Rojviboonchai. Bloom filter for fixed-size beacon in vanet. In *2014 IEEE 79th Vehicular Technology Conference (VTC Spring)*, pages 1–5, May 2014.

[9] Bo Yu and F. Bai. Pyramid: Informed content reconciliation for vehicular peer-to-peer systems. In *2015 IEEE Vehicular Networking Conference (VNC)*, pages 212–219, Dec 2015.

[10] Shirshanka Das, Alok Nandan, and Giovanni Pau. Spawn: a swarming protocol for vehicular ad-hoc wireless networks. In *Vehicular Ad Hoc Networks*, 2004.

[11] S. Panichpapiboon and W. Pattara-atikom. A review of information dissemination protocols for vehicular ad hoc networks. *IEEE Communications Surveys Tutorials*, 14(3):784–798, Third 2012.

[12] Gonçalo Pessoa, Miguel Luís, Lucas Guardalben, Susana Sargento". *On the Analysis of Content Dissemination over Wireless Access in Vehicular Environments.* IEEE 2018 IEEE 87th Vehicular Technology Conference VTC2018-Spring, Porto, Portugal, Maio, 2018.

[13] Andrei Broder and Michael Mitzenmacher. Network applications of bloom filters: A survey. *Internet Mathematics*, 1(4):485–509, 2004.

[14] P. M. Santos, J. G. P. Rodrigues, S. B. Cruz, T. Lourenço, P. M. d'Orey, Y. Luis, C. Rocha, S. Sousa, S. Crisóstomo, C. Queirós, S. Sargento, A. Aguiar, and J. Barros. Portolivinglab: an iot-based sensing platform for smart cities. *IEEE Internet of Things Journal*, PP(99):1–1, 2018.

[15] Veniam. *An internet of moving things.* [Online] Disponível: http://veniam.com, Junho, 2018.

[16] Hassnaa Moustafa and Yan Zhang. *Chapter 1: Introduction to vehicular networks.* In Vehicular Networks Techniques, Standards, and Applications, page 1-20. Auerbach Publications, Abril 2009.

[17] V. Cerf, S. Burleigh, A. Hooke, L. Torgerson, R. Durst, K. Scott, K. Fall, and H. Weiss. *Delay-Tolerant Networking Architecture,* . RFC 4838 (Informational), Internet Engineering Task Force, [Online] Disponível: http://www.ietf.org/rfc/rfc4838.txt, Junho, 2018.

[18] L. Villas. Data dissemination in vehicular networks: Challenges, solutions, and future perspectives. In *2015 7th International Conference on New Technologies, Mobility and Security (NTMS)*, pages 1–5, July 2015.

[19] Anand Paul, Naveen Chilamkurti, Alfred Daniel, and Seungmin Rho. Chapter 2 - intelligent transportation systems. In Anand Paul, Naveen Chilamkurti, Alfred Daniel, and Seungmin Rho, editors, *Intelligent Vehicular Networks and Communications*, pages 21 – 41. Elsevier, 2017.

[20] Saif Al-Sultan, Moath M. Al-Doori, Ali H. Al-Bayatti, and Hussien Zedan. A comprehensive survey on vehicular ad hoc network. *J. Netw. Comput. Appl.*, 37:380–392, January 2014.

[21] A. Dhamgaye and N. Chavhan. Survey on security challenges in vanet. *Int.J.Comput.Sci.*, 2(1):88–96, 2013. Cited By :14.

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

[23] DSRC. *dsrc.* [Online] Disponível: http://grouper.ieee.org/groups/scc32/dsrc/, Maio 2018.

[24] Kumar, R. and M. Dave. A review of various vanet data dissemination protocols. *International Journal of U-& E-Service, Science & Technology*, 2012.

[25] ETSI. *European Telecommunications Standards Institute (ETSI).* [Online] Disponível: http://www.etsi.org, Maio, 2018.

[26] Hyunwoo Kang, Syed Hassan Ahmed, Dongkyun Kim, and Yun-Su Chung. Routing protocols for vehicular delay tolerant networks: A survey. *International Journal of Distributed Sensor Networks*, 11(3):325027, 2015.

[27] NASA. *Disruption tolerant networking.* [Online] Disponível: https://www.nasa.gov/content/dtn, Maio 2018.

[28] K.R. Fall and W.R. Stevens. *TCP/IP Illustrated.* Number vol. 1 in Addison-Wesley professional computing series. Addison-Wesley, 2011.

[29] M. Ma, C. Lu, and H. Li,. *"Delay tolerant networking,"in Delay tolerant networks: Protocols and applications.* CRC press, 2011, pp. 1-29.

[30] K. Scott and S. Burleigh. *"Bundle Protocol Specification", RFC 5050 (Experimental), Internet Engineering Task Force, Nov. 2007.* [Online] Disponível: http://www.ietf.org/rfc/rfc5050.txt.

[31] S. H. Lee, U. Lee, K. W. Lee, and M. Gerla. Content distribution in vanets using network coding: The effect of disk i/o and processing o/h. In *2008 5th Annual IEEE Communications Society Conference on Sensor, Mesh and Ad Hoc Communications and Networks*, pages 117–125, June 2008.

[32] Ali Marandi, Mahdi Faghi Imani, and Kavé Salamatian. Practical Bloom filter based epidemic forwarding and congestion control in DTNs: A comparative analysis. *Computer Communications*, 48:98–110, July 2014.

[33] Y. Yi. *On-demandmulticast routing protocol (odmrp) for ad hoc networks.* IETF MANET Working Group, 2003.

[34] E. Yoneki and J. Bacon. An adaptive approach to content-based subscription in mobile ad hoc networks. In *IEEE Annual Conference on Pervasive Computing and Communications Workshops, 2004. Proceedings of the Second*, pages 92–97, March 2004.

[35] I. Parris, G. Bigwood, and T. Henderson. Privacy-enhanced social network routing in opportunistic networks. In *2010 8th IEEE International Conference on Pervasive Computing and Communications Workshops (PERCOM Workshops)*, pages 624–629, March 2010.

[36] Rafael P. Laufer, Pedro B. Velloso, and Otto Carlos M.B. Duarte. A generalized bloom filter to secure distributed network applications. *Computer Networks*, 55(8):1804 – 1819, 2011.

[37] Fabio Angius, Mario Gerla, and Giovanni Pau. Bloogo: Bloom filter based gossip algorithm for wireless ndn. In *Proceedings of the 1st ACM Workshop on Emerging Name-Oriented Mobile Networking Design - Architecture, Algorithms, and Applications*, NoM '12, pages 25–30, New York, NY, USA, 2012. ACM.

[38] Nawut Na Nakorn and Kultida Rojviboonchai. Deca: Density-aware reliable broadcasting in vehicular ad hoc networks. In *ECTI-CON2010: The 2010 ECTI International Confernce on Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology*, pages 598–602, May 2010.

[39] Jie Wu and Hailan Li. On calculating connected dominating set for efficient routing in ad hoc wireless networks. In *Proceedings of the 3rd International Workshop on Discrete Algorithms and Methods for Mobile Computing and Communications*, DIALM '99, pages 7–14, New York, NY, USA, 1999. ACM.

[40] Yu-Ting Yu, Yuanjie Li, Xingyu Ma, Wentao Shang, M. Y. Sanadidi, and M. Gerla. Scalable opportunistic vanet content routing with encounter information. In *2013 21st IEEE International Conference on Network Protocols (ICNP)*, pages 1–6, Oct 2013.

[41] Neera Batra and Rishi Pal Singh. A Hybrid Approach to Increase the Scalability in VANETs. International Journal of Applied Engineering Research 12 (16), 5729-5738. 2017.

[42] Kevin C Lee and Ian Y Yap. Cartorrent: A bit-torrent system for vehicular ad-hoc networks. University of California, Los Angeles, 05 2006.

[43] Y. Zhang, J. Zhao, and G. Cao. Roadcast: A popularity aware content sharing scheme in vanets. In *2009 29th IEEE International Conference on Distributed Computing Systems*, pages 223–230, June 2009.

[44] Hao Wu, Richard Fujimoto, Randall Guensler, and Michael Hunter. Mddv: A mobility-centric data dissemination algorithm for vehicular networks. In *Proceedings of the 1st ACM International Workshop on Vehicular Ad Hoc Networks*, VANET '04, pages 47–56, New York, NY, USA, 2004. ACM.

[45] Cristiano Rezende, Abdelhamid Mammeri, Azzedine Boukerche, and Antonio A.F. Loureiro. A receiver-based video dissemination solution for vehicular networks with content transmissions decoupled from relay node selection. *Ad Hoc Networks*, 17:1 – 17, 2014.

[46] O. K. Tonguz, N. Wisitpongphan, and F. Bai. Dv-cast: A distributed vehicular broadcast protocol for vehicular ad hoc networks. *IEEE Wireless Communications*, 17(2):47–57, April 2010.

[47] N. Wisitpongphan, O. K. Tonguz, J. S. Parikh, P. Mudalige, F. Bai, and V. Sadekar. Broadcast storm mitigation techniques in vehicular ad hoc networks. *IEEE Wireless Communications*, 14(6):84–94, December 2007.

[48] M. Bakhouya, J. Gaber, and P. Lorenz. An adaptive approach for information dissemination in vehicular ad hoc networks. *Journal of Network and Computer Applications*, 34(6):1971 – 1978, 2011. Control and Optimization over Wireless Networks.

[49] L. A. Villas, T. P. C. de Andrade, and N. L. S. da Fonseca. An efficient and robust protocol to disseminate data in highway environments with different traffic conditions. In *2014 IEEE Symposium on Computers and Communications (ISCC)*, pages 1–6, June 2014.

[50] Gökhan Korkmaz, Eylem Ekici, Füsun Özgüner, and Ümit Özgüner. Urban multi-hop broadcast protocol for inter-vehicle communication systems. In *Proceedings of the 1st ACM International Workshop on Vehicular Ad Hoc Networks*, VANET '04, pages 76–85, New York, NY, USA, 2004. ACM.

[51] E. Fasolo, A. Zanella, and M. Zorzi. An effective broadcast scheme for alert message propagation in vehicular ad hoc networks. In *2006 IEEE International Conference on Communications*, volume 9, pages 3960–3965, June 2006.

[52] Da Li, Hongyu Huang, Xu Li, Minglu Li, and Feilong Tang. A distance-based directional broadcast protocol for urban vehicular ad hoc network, 10 2007.

[53] N. Wisitpongphan, O. K. Tonguz, J. S. Parikh, P. Mudalige, F. Bai, and V. Sadekar. Broadcast storm mitigation techniques in vehicular ad hoc networks. *IEEE Wireless Communications*, 14(6):84–94, December 2007.

[54] T. Osafune, L. Lin, and M. Lenardi. Multi-hop vehicular broadcast (mhvb). In *2006 6th International Conference on ITS Telecommunications*, pages 757–760, June 2006.

[55] Qiong Yang and Lianfeng Shen. A multi-hop broadcast scheme for propagation of emergency messages in vanet. In *2010 IEEE 12th International Conference on Communication Technology*, pages 1072–1075, Nov 2010.

[56] H. Alshaer and E. Horlait. An optimized adaptive broadcast scheme for inter-vehicle communication. In *2005 IEEE 61st Vehicular Technology Conference*, volume 5, pages 2840–2844 Vol. 5, May 2005.

[57] A. Wegener, H. Hellbruck, S. Fischer, C. Schmidt, and S. Fekete. Autocast: An adaptive data dissemination protocol for traffic information systems. In *2007 IEEE 66th Vehicular Technology Conference*, pages 1947–1951, Sept 2007.

[58] Sooksan Panichpapiboon and Gianluigi Ferrari. Irresponsible forwarding. In *Proceedings - 2008 8th International Conference on Intelligent Transport System Telecommunications, ITST 2008*, pages 311 – 316, 11 2008.

[59] L. Li, R. Ramjee, M. Buddhikot, and S. Miller. Network coding-based broadcast in mobile ad-hoc networks. In *IEEE INFOCOM 2007 - 26th IEEE International Conference on Computer Communications*, pages 1739–1747, May 2007.

[60] Sachin Katti, Hariharan Rahul, Wenjun Hu, Dina Katabi, Muriel Médard, and Jon Crowcroft. Xors in the air: Practical wireless network coding. *SIGCOMM Comput. Commun. Rev.*, 36(4):243–254, August 2006.

[61] N. Kadi and K. A. Agha. Mpr-based flooding with distributed fountain network coding. In *2010 The 9th IFIP Annual Mediterranean Ad Hoc Networking Workshop (Med-Hoc-Net)*, pages 1–5, June 2010.

[62] Prithwish Basu and Chi-Kin Chau. Opportunistic forwarding in wireless networks with duty cycling. In *Proceedings of the Third ACM Workshop on Challenged Networks*, CHANTS '08, pages 19–26, New York, NY, USA, 2008. ACM.

[63] G. Pessoa, R. Dias, T. Condeixa, J. Azevedo, L. Guardalben, and S. Sargento. New insights on content distribution emulation for vehicular networks. In *The Wireless Days Conference 2017*, March 2017.

[64] D. Ho. T.; Lun. Network coding: An introducion. In *Cambridge University Press: New York, NY, USA*, 2008.

[65] iMatix Corporation. *Zmq.* [Online] Disponível: http://zeromq.org/, Março, 2018.

[66] L. Guedes. *Transmissão Oportunística de Informação em Redes Veiculares.* Master's Thesis, Departamento de Engenharia Eletrónica Telecomunicações e Informática, Universidade de Aveiro, 2014.

[67] A. Partow. *General purpose hash function algorithms.* [Online] Disponível: http://www.partow.net/programming/bloomfilter/index.html, Março, 2018.

[68] OpenWRT. *Openwrt - wireless freedom.* [Online] Disponível: http://www.openwrt.org/, Junho, 2018.

[69] OpenWRT. *Openwrt build system.* [Online] Disponível: http://wiki.openwrt.org/doc/howto/build, Junho, 2018.

[70] Network Time Foundation. *Ntp: The network time protocol.* [Online] Disponível: http://www.ntp.org/, Junho, 2018.

[71] FutureCities Project. Living lab: Vehicular adhoc networking, http://futurecities.up.pt/site/vehicular-ad-hoc-networking-testbed/, 2015.

[72] Matlab. *Mathworks.* [Online] Disponível: /http://www.mathworks.com/products/matlab/, Junho, 2018.

Apêndice A

mOVERS Source Code NetRider Integration

Este apêndice encontra-se em inglês para servir de base para utilização a nível internacional.

The following appendix serves as a guide on how to compile source code from mOVERS to NetRider version 3.

A.1 Compiling

First, it is required the sdk from `http://wiki.openwrt.org/doc/howto/build` and then, after downloading the project from `https://code.nap.av.it.pt/CDN-VANETS/DD_Filters/tree/master/mOVERS_NetRiders` (this project was cleaned and adapted exclusively only to be used in cross-compiling) and making the necessary modifications on what it is required to test, run the following commands inside the mOVERS folder:

mkdir build
cd build
cmake -DCMAKE_TOOLCHAIN_FILE= /[full_path]/sdk/toolchain-ar71xx.cmake ..
make

The generated binary (called movers) is ready to transfer to the NetRiders boards and is in **build/bin**. Once the binary if transferred to the boards, create the **api, log, storage** folders at the same level. Also create the file named **type** and write obu/rsu as its content depend on the node to test. Create the file named **dateRef.txt** and write the atual unix timestamp in it to be used to save the log file. The file **helix.conf** located in **movers/config/** also needs to be transferred, placed at the same level and needs to be according with the following:

```
{
        "socket_port": "4556",
        "storage_path": "storage/",
        "storage_cap": "5",
        "api_path": "api",
        "log_path": "log",
```

```
        "log_output": "2",
        "log_level": "3",
        "11p_iface": [ "wlan1" ],
        "11g_iface": [ "" ],
        "rt_version": <your_routing_version>
}
```

It is important do not forget that all paths defined in this *json* must exist in the board; otherwise, an error will occur at the start of mOVERS.

A.2 Synchronization

To synchronize each of the boards, first run the following command on the board aimed to be the RSU, acting as a server:

ntpd -l

Then, run the following command in each of the boards aimed to be OBUs, acting as clients:

ntpd -q -p [RSU_IP]

Note: [RSU_IP] can be the IP of either Ethernet or WAVE interface. To confirm that the boards are in fact synchronized, the following command can be used in each of them:

date

A.3 Running

To run mOVERS in the boards for a content distribution test, the following command should de executed at the same level:

./movers -c -f helix.conf

Or execute the following command to execute in background (in this case the folder **nohup_files** must be created before):

./movers -c -f helix.conf > nohup_files/[type_node_and_id.out] 2>&1 &

A.4 Problems that may occur

One of the most problems that may occur when running mOVERS in boards is the lack of some libraries. The most lacking library is **libjson-c.so.2**. If this happen it is need to install it in the board. A quick search in url `http://archive.openwrt.org/barrier_breaker/14.07/ar71xx/mikrotik/packages/base/` provides the missing library.