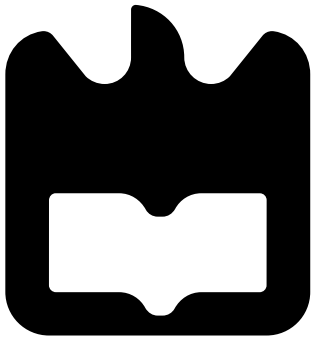
Universidade de Aveiro
Departamento de
Electrónica, Telecomunicações e Informática,
2016

Daniel
Vicente
Inácio

Distribuição de Conteúdos em Redes Veiculares com mecanismos de filtragem

Content Distribution in Vehicular Networks with filtering mechanisms



Daniel
Vicente
Inácio

Distribuição de Conteúdos em Redes Veiculares com mecanismos de filtragem

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Engenharia Electrónica e Telecomunicações, realizada sob a orientação científica da Professora Doutora Susana Sargento, Professora do Departamento de Electrónica, Telecomunicações e Informática da Universidade de Aveiro e do Doutor Ricardo Matos, Director de IP & Compliance na Veniam.

o júri / the jury

presidente / president

Professor Doutor Nuno Miguel Gonçalves Borges de Carvalho
Professor Catedrático 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, Universidade de Aveiro (orientadora)

Professora Doutora Marília Pascoal Curado
Professora Auxiliar da Universidade de Coimbra

Dedication

I would like to take this opportunity to thank my parents and my sister and my grandparents for their unrelentless support throughout my academic degree.

I would also like to show my appreciation to the professor Susana Sargento for captivating me into the area of Telecommunications during my fourth year of the degree as well as for the opportunity to develop a Dissertation under her guidance and allowing my integration into the NAP group.

Last but far from least, I would like to thank all the friends and colleagues who have accompanied me throughout the degree as well as those I've met in the Instituto de Telecomunicações, specifically, Lucas Guardalben, Bruno Areias, André Reis and Carlos Ferreira for their constant patience, availability and aid in the development of this Dissertation.

Resumo

Conectividade representa uma grande necessidade da população desde o início dos tempos. As pessoas têm, logo à partida, um desejo de estarem ligadas entre si e ao resto do mundo. Tal não mudou nos tempos actuais, especialmente na era das novas tecnologias onde conectar-se com alguém está apenas a uns cliques de distância. Do ponto de vista de engenheiros da área das telecomunicações, este rápido desenvolvimento nas comunicações sem fios tem sido especialmente marcante.

Devido a esta constante necessidade de comunicação, as VANETs (Vehicular Ad-Hoc Networks) atraem actualmente um interesse significativo. Esse interesse deve-se ao facto de as redes veiculares não só poderem ser usadas para uma condução potencialmente mais segura, como também poderem proporcionar aos passageiros o acesso à Internet.

As redes veiculares têm características específicas face a outro tipo de redes, tais como o número elevado de veículos ou nós, rotas imprevisíveis e a constante perda de conectividade entre os mesmos, revelando vários desafios que propõem estudos para os solucionar. A solução encontrada para a conectividade intermitente prende-se com o uso de DTNs (Delay-Tolerant Networks) cuja arquitectura assegura a entrega de informação mesmo quando não há conhecimento do percurso completo que esta deve percorrer.

Esta Dissertação de Mestrado foca-se no estudo da disseminação de conteúdo não-urgente via uso de DTNs, assegurando que esta mesma disseminação é feita no menor espaço de tempo possível e com o mínimo congestionamento possível na rede. Actualmente, embora a entrega de informação já seja efectuada na rede num espaço de tempo satisfatório, as estratégias implementadas forçam um congestionamento (*overhead*) considerável na rede. Para combater este efeito, foi desenvolvida uma estratégia de disseminação através do uso de *Bloom Filters*, uma estrutura de dados capaz de eliminar a maior parte dos acessos desnecessários à memória, assegurando a um nó a existência de um pacote específico, com uma certa probabilidade, de entre toda a informação que os seus vizinhos contêm.

Esta estratégia foi implementada no *software* de DTNs mOVERS Emulator, desenvolvido pelo Instituto de Telecomunicações de Aveiro (IT) e pela Veniam® e posteriormente testada no mesmo. O emulador utilizado simula uma rede veicular com base em informação recolhida da rede veicular da cidade do Porto.

Após análise dos resultados obtidos, foi concluído que a nova estratégia de disseminação proposta, denominada FILTER, cumpriu o principal objectivo proposto, nomeadamente, a redução do *overhead* na rede veicular, com uma pequena perda de taxa de entrega da informação. Para trabalho futuro, é aconselhável realizar um estudo mais extenso em métodos relacionados com utilidade da informação para optimizar essa mesma taxa de entrega.

Abstract

Connectivity represents one of people's great needs since the beginning of times. From the start, people have a desire to be connected to each other and to the rest of the world. Such has not changed in modern times, especially in the era of new technologies where connecting with someone is only a few clicks away. From the point of view of engineers in the area of telecommunications, this fast development in wireless communications has been especially outstanding.

Due to this constant need for communication, VANETs (Vehicular Ad-Hoc Networks) are currently attracting significant attention. Such attention is due to the fact that vehicular networks may be used for, not only potentially safer driving, they also provide its users with Internet access.

Vehicular Networks have specific characteristics when compared to other types of networks, such as the high number of vehicles or nodes, unpredictable routes and the constant loss of connectivity between these nodes, thus revealing several challenges which propose studies to solve them. The solution found for the intermittent connectivity involves the use of DTNs (Delay-Tolerant Networks) whose architecture ensures the delivery of information even without knowledge of the whole path it must travel.

This Masters Dissertation focuses on the study of non-urgent content dissemination through the use of DTNs, ensuring that this same dissemination is done within the shortest time frame and with the minimum congestion possible in the network. Currently, though the information delivery is already performed in the network with a satisfactory time frame, the implemented strategies force considerable congestion in the network. To overcome this effect, a dissemination strategy was developed through the use of Bloom Filters, a data structure capable of eliminating most of the unnecessary access to memory, by ensuring a node the existence of a specific packet, with a certain probability, from among all the information its neighbours contain.

This strategy was implemented in the DTN software mOVERS, developed by Instituto de Telecomunicações in Aveiro (IT) and Veniam® and posteriorly tested in the same emulator. The emulator used simulates a vehicular network with information gathered from the vehicular network in the city of Porto.

After the analysis of the obtained results, it was concluded that the new proposed dissemination strategy, named FILTER, has fulfilled its primary objective, namely, the reduction of the vehicular network's *overhead*, with a small loss in the delivery rate of the information. For future work, it is advisable to perform a more extensive study in methods related to the information's usefulness to a neighbour to optimize such delivery rate.

Contents

List of Figures

List of Tables

Acronyms

ACK	Acknowledgement
ADV	Advertisement
AODV	Ad-Hoc On-Demand Distance Vector
AP	Access Point
API	Application Programming Interface
AU	Application Unit
BSS	Basic Service System
C2C-CC	Car-to-Car Communication Consortium
CLA	Convergence Layer Adapter
COOPERS	Cooperative Systems for Intelligent Road Safety
CPU	Central Processing Unit
DTN	Delay-Tolerant Network
EID	Endpoint Identifier
GPS	Global Positioning System
HUNET	Human Network
IEEE	Institute of Electrical and Electronics Engineers
IP	Internet Protocol
IPC	Inter-Process Communication
IT	Instituto de Telecomunicações
LNHF	Least Number of Hops First

LRBF Local Rarest Bundle First

LRGF Local Rarest Generation First

LTE Long Term Evolution

MAC Medium Access Control

MANET Mobile Ad-Hoc Network

MATLAB MATrix LABoratory

MDDV Mobility-Centric Data Dissemination

mOVERS mobile Opportunistic Vehicular Emulator for Real Scenarios

NASA National Aeronautics and Space Administration

NDN Named Data Networking

NoW Network on Wheels

NS Network Simulator

OBU On-Board Unit

ODMRP On-Demand Multicast Routing Protocol

ONE Opportunistic Network Environment

OSNR Obfuscated Social Network Routing

OSI Open Systems Interconnection (Model)

P2P Peer-to-Peer

RAM Random-Access Memory

REDEC REceiver-based solution with video transmission DECoupled

RSSI Received Signal Strength Indicator

RSU Road Side Unit

SCF Store-Carry-and-Forward

SUMO Simulation Urban MObility

SPAWN Swarming Protocol For Vehicular Ad-Hoc Wireless Networks

SCP Store-Carry-and-Forward

TCP	Transmission Control Protocol
UDP	User Datagram Protocol
UNIX	Uniplexed Information Computing System
V2I	Vehicle-to-Infrastructure
V2V	Vehicle-to-Vehicle
VANET	Vehicular Ad-Hoc Network
VDTN	Vehicular Delay-Tolerant Network
WAVE	Wireless Access for Vehicular Environments
Wi-Fi	Wireless Fidelity

Chapter 1

Introduction

This document was developed in the scope of the Masters Dissertation in Electronics and Telecommunications Engineering in the Department of Electronics, Telecommunications and Informatics at the Universidade de Aveiro, with the theme of "Content Distribution in Vehicular Networks".

In this chapter, both context and motivation for this Dissertation are presented, along with its objectives and contributions. It also briefly describes this document's structure.

1.1 Context and Motivation

Since very early on, the act of communicating with one another has always been of vital importance in society. However, the evolution in communication, specifically in Telecommunications, has grown at a particularly fast pace, to the point that we now have more things connected to the Internet than the number of people in the world. And that number is expected to keep growing until, at least, 2020 [1].

The automotive market is growing slowly but steadily and it is expected to see more than 100 million passenger cars by 2018 [2]. Such advancement in automotive technology is due to electronics playing a major role, offering constant innovations, from the well-known Global Positioning System (GPS) and automatic parking to advanced safety features, energy efficiency and new information and entertainment services.

As a result of these two growing markets, and since users enjoy being connected and able to access content at any time, it is expected in the future to have all vehicles becoming nodes in the network, having early access to non-urgent content such as weather reports or road traffic, as well as emergency content such as traffic accidents. Current testing and product trials are already suggesting vehicle-to-vehicle communications to reduce the chance of collisions [3]. Veniam® [4], along with the University of Aveiro, University of Porto and the Instituto de Telecomunicações (IT), lead to the implementation of the largest vehicular network, worldwide, with over 600 vehicles (buses, garbage trucks, taxis, among others) and a fixed infrastructure, located in the city of Oporto.

Communication between vehicles imposes challenges due to the high mobility of the

nodes (vehicles) and the dispersion of the same. As a result, vehicular ad-hoc networks must use mechanisms capable of tolerating intermittent connectivity and long delay. The type of architecture capable of transmitting information reliably in this environment has already been developed and is called Delay-Tolerant Network (DTN) [5].

The purpose for using these networks composed of moving vehicles is to have all of these nodes working together, sharing content among each other by sending to their neighboring vehicles information that they do not have. Figure 1.1 illustrates a typical VANET. All the vehicles within range of the infrastructure receive information that is then shared among all neighboring vehicles within range.

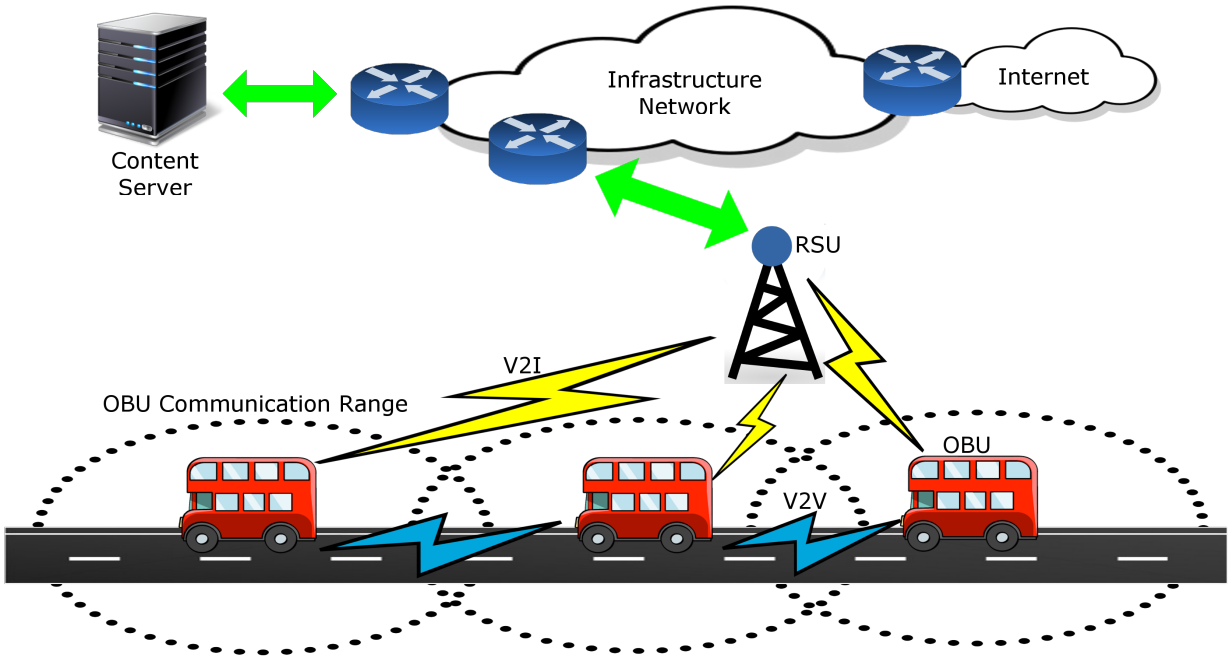


Figure 1.1: VANET Architecture

Due to the challenges associated with a highly mobile network, there are limitations as to the kind of information to be sent. Preferred services should be those that do not require immediate transmission nor a constantly stable end-to-end path, such as tourism-related information, advertisements and sensory data (non-urgent content). Another type of preferred services are those that must have a higher priority in transmission, such as emergency services (urgent content). However, while disseminating this information, other challenges have to be faced. Depending on the algorithm responsible for deciding how to distribute content among the nodes, extra information regarding control may need to be sent along with the actual data in order to ensure that packets are being distributed with maximum efficiency. This control information is called the *overhead* and, if the algorithm is heavily focused on controlling how the information is distributed, the network may end up "clogged" with information unrelated to what the destination nodes want. Therefore, a content dissemination

strategy must be able to distribute content to the highest number of nodes possible in the least amount of time while minimizing the network overhead.

This is the framework of this Dissertation. Focused on using DTNs for disseminating non-urgent information while ensuring the minimum amount of control packets possible. In order to do so, a DTN solution developed by IT and Veniam® named mobile Opportunistic Vehicular Emulator for Real Scenarios (mOVERS) is used in a simulator called *mOVERS Emulator* in order to analyze the effects of a dissemination strategy with reduced overhead in a Vehicular Ad-Hoc Network.

1.2 Objectives and Contributions

The main purpose of this Dissertation is the study and implementation of a content dissemination strategy through a vehicular ad-hoc network with minimum network overhead. As such, the objectives are as follows:

- Survey of related work;
- Study of the existent platforms of development;
- Statistical analysis of the real vehicular network;
- Study of previously implemented content dissemination strategies;
- Design and implementation of a strategy focused on reducing network overhead;
- Tests and analysis of the implemented strategy.

This Dissertation has provided the following contributions:

- While studying previously obtained data from the vehicular network, a statistical analysis of this data allowed for proper understanding of which timeframes of study would be more relevant;
- Real experiments were performed with previously implemented content distribution strategies to more easily understand the vehicles' behaviour in the network.
- A new content distribution strategy with reduced network overhead has been proposed, implemented and evaluated. A new module and some global modifications were made to the mOVERS emulator;
- A paper with this approach is being prepared to be submitted in an international conference.

1.3 Document Structure

- **Chapter 1 - Introduction:** Contextualizes the Dissertation, presenting its motivation, proposed objectives and document structure;
- **Chapter 2 - State of the Art:** Provides an overview of the main concepts of Vehicular Ad-hoc Networks, Delay-Tolerant Networks and a survey on previously created content dissemination strategies;
- **Chapter 3 - Content Distribution Strategy:** Provides the scope of the previously implemented dissemination schemes and presents the proposed distribution strategy;
- **Chapter 4 - Integration and Development:** Describes the design and implementation of the aforementioned strategy in the development platform for posterior evaluation;
- **Chapter 5 - Tests and Results:** Describes the several evaluated scenarios and its results, along with its evaluation and discussion;
- **Chapter 6 - Conclusions and Future Work:** Discusses the conclusions of the developed work and proposes future improvements for continued research;

1.4 Summary

This chapter presented the motivation for this Dissertation, as well as its primary objectives and a brief description of its document structure. The next chapter will provide the reader with a range of general information regarding Vehicular Ad-Hoc Networks and Delay-Tolerant Networks.

Chapter 2

State of the Art

2.1 Chapter Description

This chapter provides insight in the topics of relevance regarding the work developed, pursuing this degree, using published literature. The organization of this chapter is as follows.

Section 2.2 presents the basic concept and definition of a VANET and its characteristics, as well as its architecture, challenges and applications.

Section 2.3 introduces the concept of a DTN, its architecture and applications.

Section 2.4 clarifies the concept of using a DTN in a VANET, henceforth called Vehicular Delay-Tolerant Network (VDTN)s, as well as its applications.

Section 2.5 describes several content distribution solutions used in many different projects. After a small survey on peer-to-peer schemes, a more focused study on Bloom Filters is provided.

Section 2.6 depicts a set of network, VANET and DTN simulators, used to evaluate network and routing protocols of networks.

Section 2.7 summarizes this full chapter.

2.2 Vehicular Ad-Hoc Networks

2.2.1 Definition

Due to the growth in both the automotive industry and wireless technology industry [1][2], these advances have led to a new class of wireless networks called Vehicular Ad-Hoc Networks, also known as VANETs. This type of networks is formed by a set of moving vehicles equipped with wireless interfaces, enabling communication among each other and with the infrastructure used in this network.

These vehicles, from now on referred to as nodes, have to be equipped with On-Board Units (OBUs), to communicate with each other or with the infrastructure via fixed equipments placed on the roads called Road Side Units (RSUs). In a VANET, since OBUs are mobile nodes, they are responsible for the dissemination of information throughout the network. On the

other hand, RSUs are fixed nodes, responsible for allowing access to outer networks.

As a result, communication can be classified in a VANET in two different types: communication between nearby vehicles (Vehicle-to-Vehicle (V2V)) and between vehicles and the infrastructure (Vehicle-to-Infrastructure (V2I)).

2.2.2 Characteristics

Vehicular Networks are a type of Mobile Ad-Hoc Networks (MANETs) but have inherent characteristics that distinguish them from other types of mobile networks [6].

- **Unlimited Transmission Power:** Since nodes are vehicles, they have the capability to supply continuous power to computing and communication devices.
- **Higher Computational Capacity:** Due to the mentioned unlimited power, the nodes can afford significant computing, communication and sensing capabilities.
- **Predictable Mobility:** Vehicles tend to have, to an extent, predictable movement due to being constrained by roads. Currently available technologies, such as GPS, already have roadway information available and, given the road trajectory as well as average and current speeds of a vehicle, it is possible to predict the future position of a node.

2.2.3 Challenges

Vehicular Ad-Hoc Networks are capable of supporting a diverse range of applications and services. Therefore, they require effective resource management strategies. However, since VANETs are different from other mobile networks, the challenges inherent to them require specific strategies. The challenges of Vehicular Ad-Hoc Networks are as follows [6][7]:

- **Potentially Large Scale:** Vehicular Networks can extend over the entire road network, including many participants, potentially extending around the globe.
- **Dynamic Topology/High Mobility:** Although the nodes' movement is somewhat restricted to roads and allow route prediction to an extent, vehicles move at varying speeds, especially in urban areas, leading to a constantly changing topology.
- **Intermittent Connectivity:** Due to the two above-mentioned challenges, link connectivity also changes frequently in VANETs. Especially when considering highways, where the density of vehicles is lower when compared to urban areas.
- **Heterogeneity of Applications:** VANETs must be capable of providing a wide range of road safety and informative applications. Road safety applications, being emergency content, require low delay and high priority. On the other hand, informative applications require better throughput and higher resource utilization but can afford higher delay. Devising an approach to ensure both services in a highly dynamic environment is necessary.

2.2.4 Equipment

The constant enhancements in wireless technologies have already allowed real implementations of VANETs by equipping vehicles and roads with the appropriate equipment. A VANET's architecture uses three main components: an OBU, Application Unit (AU) and RSU [8].

The OBU is a device mounted on-board a vehicle and is used to communicate with other OBUs or RSUs. It is equipped with a Central Processing Unit (CPU) for command processing, a read/write memory for information storage and retrieval, a user interface and a network device for wireless communications, using the Wireless Access for Vehicular Environments (WAVE) technology. They can also be equipped with other interfaces, such as Wireless Fidelity (Wi-Fi) or Cellular.

The AU is responsible for executing applications, making use of the OBU's communication capabilities. These components can be connected through a wireless or wired interface and, therefore, this unit can be a dedicated device for safety applications or a personal device, such as a smartphone.

The RSU is a fixed device, typically along the road side or in dedicated locations, such as a parking lot. It is also equipped with network devices, using WAVE and can also be equipped with other interfaces for the purpose of communicating with the infrastructure. These units are responsible for extending the range of the network, disseminating information to the OBUs and running safety applications, acting as an information source.

Figure 2.1 shows examples of these devices.

(a) OBU (b) RSU

Figure 2.1: On-Board and Road Side Units

2.2.5 WAVE Protocol

In order to address the specific needs of VANETs, Institute of Electrical and Electronics Engineers (IEEE) developed standards for Wireless Access in Vehicular Environments, the WAVE Protocol, specifically designed for communications between nodes in Vehicular Networks. These standards are composed by the IEEE 802.11p and the IEEE 1609.X family [9].

IEEE 802.11p

Due to the several requirements imposed by vehicular networks, IEEE developed 802.11p, a standard capable of working in vehicular environments by providing a set of functions and services which can be used in high speed environments, where the physical layer properties change rapidly and communications have short duration. In this manner, there is no need to join a Basic Service System (BSS)) to exchange messages (unlike the traditional IEEE 802.11). Moreover, this protocol also defines the interface functions and signaling, controlled by the Medium Access Control (MAC) layer [10].

1609.X

The 1609.X family is composed of four trial use standards, each one specifying different services and applications provided by the protocol. These standards are described below:

- **IEEE 1609.1:** Provides a resource manager to the WAVE architecture, defining the key elements of the VANET, namely services, interfaces, data storage formats and command message formats for the interaction between network equipments;
- **IEEE 1609.2:** Defines secure message formats, the circumstances for using them and how to process them;
- **IEEE 1609.3:** Defines the network and transport layer services, including addressing and routing, to support secure WAVE data exchange;
- **IEEE 1609.4:** Provides enhancements to the IEEE 802.11 MAC layer to support multi-channel operations.

As observed in Figure 2.2 below, IEEE 802.11p is used in the physical and MAC layers while the 1609.X family deals with all the upper layers and the MAC layer as well.

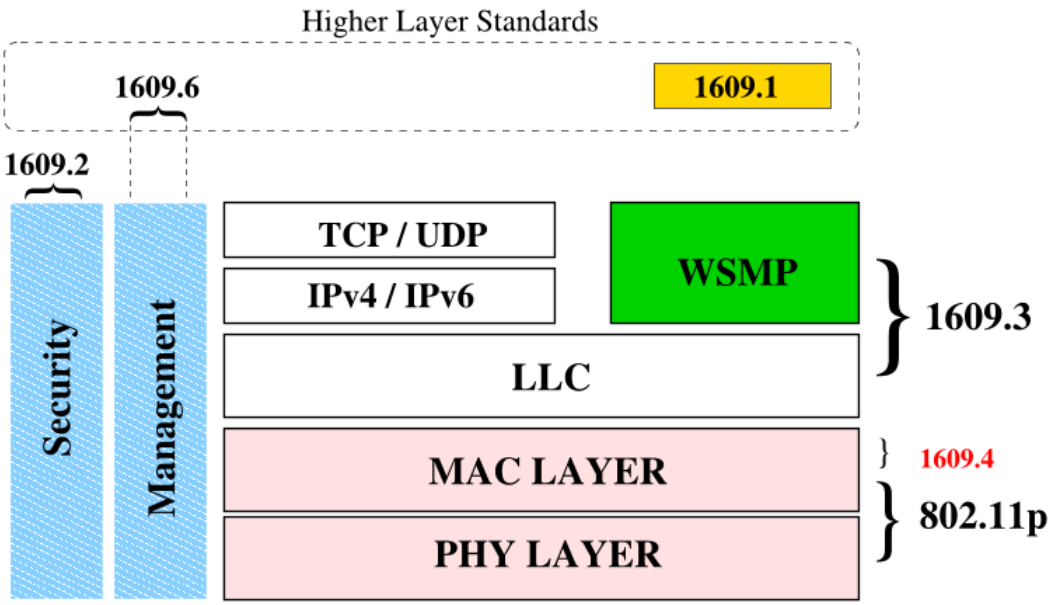


Figure 2.2: WAVE Stack [11].

2.2.6 Dissemination of Information

Most applications in VANETs involve transmitting data between the network's nodes, both OBUs and RSUs. Since VANETs operate in environments where node density and connectivity intermittency can vary, these networks require routing algorithms that ensure correct data dissemination. As observed in Figure 2.3, data dissemination can be classified according to the number of hops that a message has to travel, single-hop or multi-hop.

Data dissemination using **single-hop** is usually implemented with broadcast on the MAC layer. Vehicle A sends information to all neighbours within its range. In this case, Vehicle B is out of vehicle A's range and, therefore, cannot receive the information.

Data dissemination using **multi-hop** is more similar to VANETs with data being transmitted in several hops, using intermediate vehicles to receive and then send the message to the next node. Through this method, Vehicle B can now receive Vehicle A's message by using Vehicle C as an intermediary.

Both these systems can be combined in a hybrid variant. For example, information could be sent using multi-hop to the nearest RSU which would then use single-hop to share the information with passing vehicles.

Data can also be disseminated, taking into account the number of destinations, being classified as unicast, multicast or broadcast. Data dissemination in **unicast** involves only one sender and one receiver. Most messages disseminated in unicast are related to entertainment applications, such as Internet access, games, videos, among others [12].

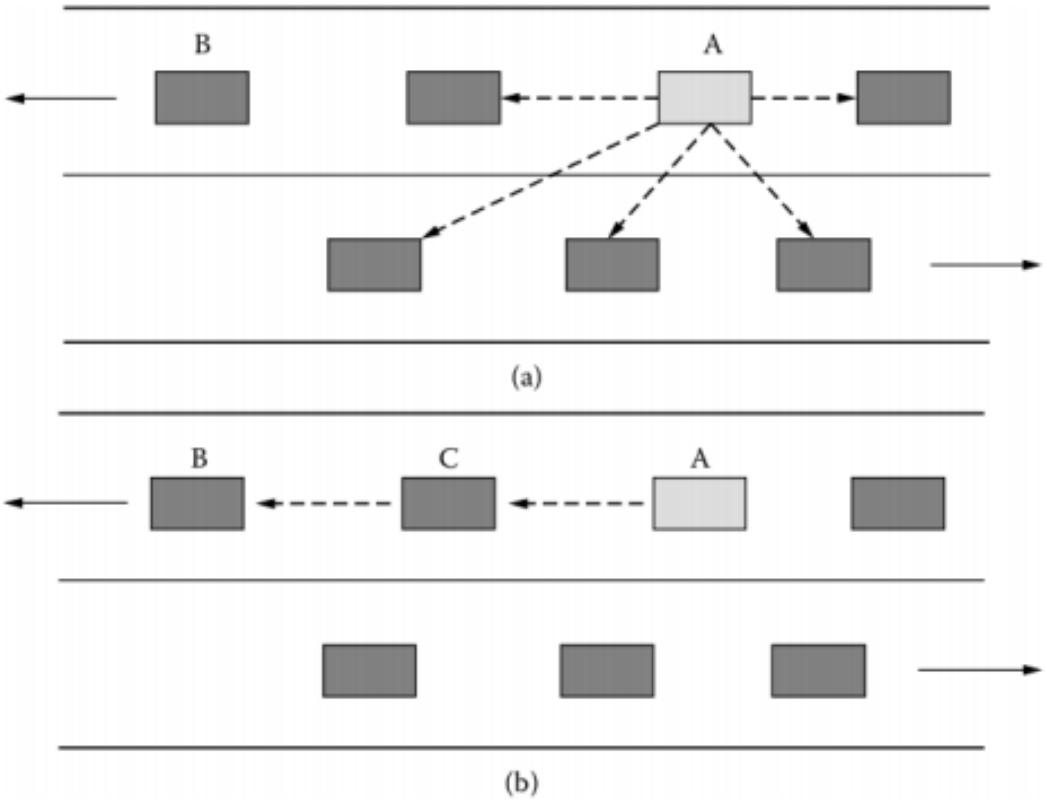


Figure 2.3: Single-Hop and Multi-Hop Data Dissemination [6].

Data dissemination in **multicast** involves one sender and several receivers. Safety applications can be used in this context. For example, only vehicles within the radius of a traffic accident require such information, making those nodes a multicast group.

Data dissemination in **broadcast** involves one or more senders sending their data to all vehicles within their range. This is the context of our work in this Dissertation.

2.3 Delay-Tolerant Networks

2.3.1 Definition

Delay-Tolerant Networks, labeled DTNs, are defined by NASA as "a computer networking model and a system of rules for transmitting information when the delay and potential for disruption or data loss is significant" [13]. This type of network was created, primarily to extend terrestrial Internet capabilities, allowing for space communications, an environment subject to frequent communication intermittency, long delays and high error rates.

Since this type of network protocol was developed, taking such environments into ac-

count, DTNs are far more flexible and can adapt to other extreme environments other than space. Upon publishing of this architecture's first draft [14], the authors aimed to use the DTN architecture in Earth's wireless networks for opportunistic routing.

2.3.2 Architecture

Store-Carry-and-Forward Mechanism

DTNs are characterized for their ability to provide communication between entities when connectivity is poor due to the network's environment. In order to ensure that a reliable link is provided between nodes, the issues of high error rate, long delays and intermittent connectivity need to be addressed. The mechanism which allows DTNs to perform in such environments is called the Store-Carry-and-Forward mechanism. DTNs, therefore, receive the data, store it and are only able to forward it when communication is available.

Depending on the type of network where DTNs are used, communication can be unavailable to a node for a considerable amount of time (hours or days). Moreover, if the communication link disrupts during transmission of information, such information must still be stored in the sender node for posterior retransmission. As a result, nodes need to be equipped with hard drives for storage purposes.

In conclusion, the Store-Carry-and-Forward mechanism is the answer to intermittent connectivity. An illustration of this mechanism is provided in Figure 2.4.

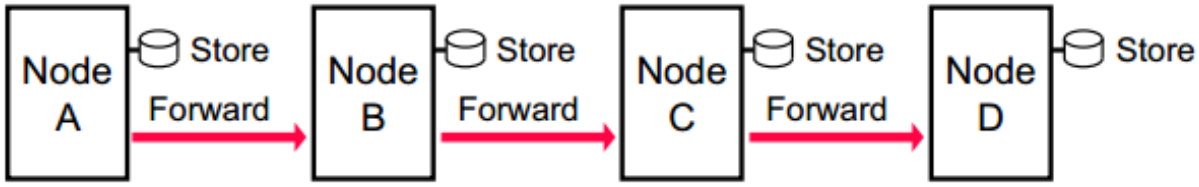


Figure 2.4: Store-Carry-and-Forward Mechanism [15].

Overlay Architecture

In order to include the Store-Carry-and-Forward mechanism in the network, the DTN architecture adds a new layer between the Application and Transport Layers. According to [14], the bundle layer is responsible for employing persistent storage as a countermeasure to intermittent connectivity, as well as the Store-Carry-and-Forward mechanism. It also includes other functionalities, such as optional end-to-end acknowledgment, diagnostic and management features and a basic security model to prevent infrastructure's unauthorized use. Figure 2.5 represents the OSI Model with the Bundle Layer integrated.

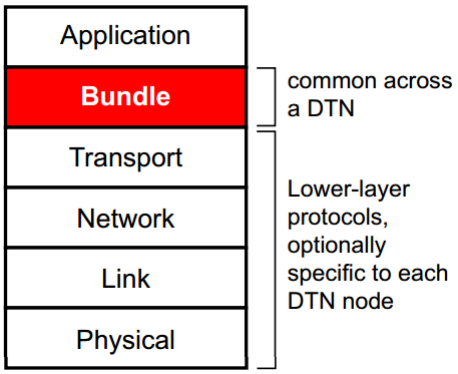


Figure 2.5: DTN Protocol Stack [15].

A DTN can use several different protocols for data delivery, from TCP/IP protocol to raw Ethernet and serial lines. Since each protocol has its own conventions and semantics, the DTN Architecture includes a Convergence Layer Adapter, or CLA, responsible for providing all the functions required to carry DTN data bundles to their corresponding protocol.

As observed in Figure 2.6, the DTN Architecture's central element is the Bundle Forwarder, responsible for forwarding bundles between storage, the CLA and the applications, depending on the decisions made by routing algorithms. Aside from bundles, this forwarder also exchanges directives used by the management entity, the applications or the routing algorithm.

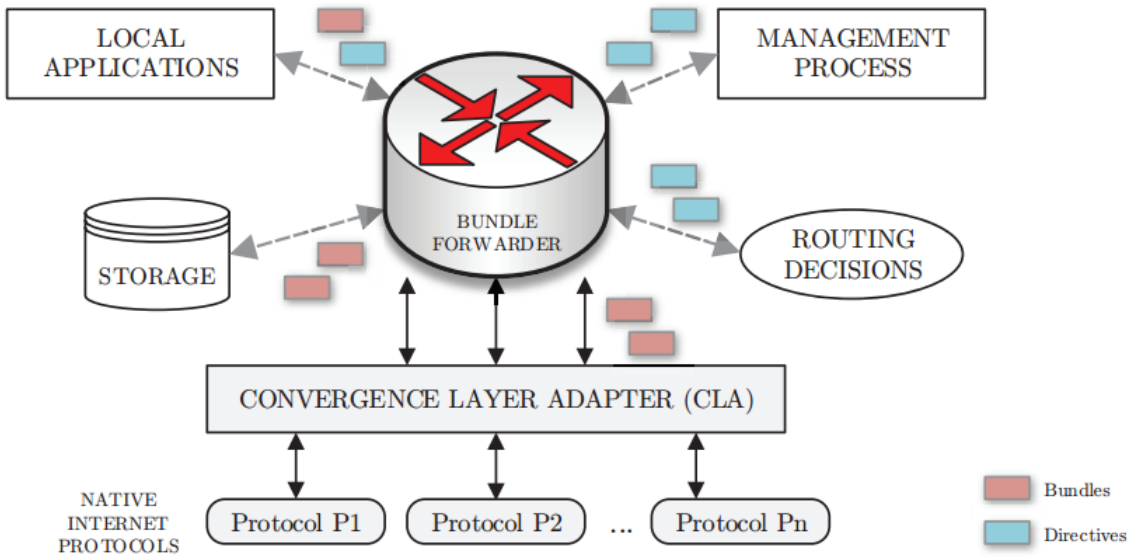


Figure 2.6: Bundle Forwarder Architecture [16].

2.4 Vehicular Delay-Tolerant Networks

Vehicular networks are self-organized by vehicles, responsible for routing packets to each other. However, due to the vehicles' high mobility and intermittent connectivity, network partition is a common occurrence in VANETs, especially when node density is sparse. As a result, it is not easy to establish a robust end-to-end path between a packet's source node and destination node, even though most VANET projects assume there is one.

On the other hand, DTNs use a communication protocol that already assumes there is no end-to-end path between source and destination and accept the notion of a frequently-partitioned network.

The characteristics of DTNs make them suitable for VANETs. As such, vehicular networks can be treated as delay-tolerant networks and will, henceforth, be called Vehicular Delay-Tolerant Networks or VDTNs.

2.4.1 Applications

VDTNs have a wide range of applications and in [17], a survey was conducted, listing some of those uses for vehicular networks, those being the following:

- **Notification of Traffic Conditions:** Sending warnings to vehicles regarding traffic jams to optimize traffic flow;
- **Weather Reports:** Warning drivers about bad conditions of the road, due to fog, ice or wind, for example;
- **Advertisements:** Enabling tourists access to relevant information, multimedia files, parking spots, among others;
- **Internet Access:** Also for entertainment purposes, i.e., enabling users access to e-mail or web browsing.

2.4.2 Vehicular Delay-Tolerant Network Projects

From early developments focused on developing the WAVE Protocol to current field trials using real-life VANET implementations, a great deal of effort has already been invested in vehicular networks' systems around the world and these trials continue to this day. In [17][18], surveys on vehicular network projects have already been conducted.

The following subsections will describe a number of those Research & Development projects as well as their main characteristics.

NoW - Network on Wheels

Network on Wheels (NoW) is a German joint project founded by several automobile manufacturers and supported by the Federal Ministry of Education and Research [19]. This project

supports the integration of both safety and non-safety applications in a communication system, providing an open platform for a broad spectrum of applications.

PreVENT

PreVENT is a European Union sponsored project to contribute to road safety by developing and testing safety applications [20]. This project evaluated preventive safety measures in several different situations such as measuring safe following distance, safe speed driving, collision mitigation, among others.

KioskNet

The KioskNet project is a system developed by the University of Waterloo, designed to provide low cost Internet access to rural areas (kiosks) in developing countries [21]. Only a few services were provided, namely e-mail, web browsing, telemedicine, among a few others.

Due to the limitations of the area, these kiosks have no permanent Internet connection and, therefore, a bus and DTN protocols provide a gateway between the kiosks and a neighboring town. The user applications generate bundles, kept in storage until the bus passes by, collecting them and later delivering them to an Internet gateway, as observed in Figure 2.7.

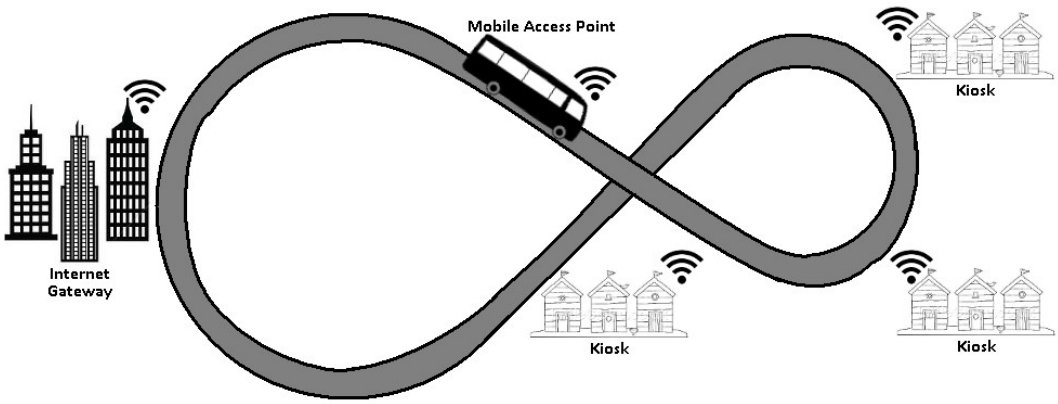


Figure 2.7: KioskNet Schematic, based on [21].

COOPERS

Cooperative Systems for Intelligent Road Safety (COOPERS) focused on developing telematics applications on the road infrastructure in order to create a "Cooperative Traffic Management" between vehicle and infrastructure. The goal of the project was to enhance road safety using traffic information obtained through road sensors. It demonstrated results at major European motorways with high-density traffic [6][22].

CarTel

CarTel is a mobile sensor computing system designed to collect and deliver data from sensors located in vehicles [23]. Each sensor is responsible for gathering readings and delivering them to a central point that stores the data into a database for further analysis. This project runs on a small scale in Boston and Seattle, having been deployed in six cars to analyze commute times, metropolitan Wi-Fi deployments and automotive diagnostics.

Drive-Thru Internet

The Drive-Thru Internet project provides Internet access to vehicles by exploiting intermittent connectivity to road access points. In order to enable useful communications, even in cases of extreme intermittent connectivity, an intermediary *proxy* is introduced in the fixed road stations, "shielding" a certain application's clients and servers from the intermittent connectivity [24].

DieselNet

Simulators often assume variables such as the On-Board Units' power consumption, mobility routes, communication radius, among others. As a result, simulation can differ greatly from a real-life implementation. The DieselNet project was designed to overcome simulation-related problems of mobile networks and is composed of three major components: the vehicular network, nomadic throwboxes and an outdoor mesh network [25].

The vehicular network is comprised of 40 buses travelling across urban and rural areas, covering an area of 150 square miles. Throwboxes are nodes acting as relays to create extra contact opportunities. Though they often remain stationary, they can be placed anywhere on the network, providing flexibility to node connectivity. The mesh network is composed of 26 Wi-Fi access points, providing a direct connection to the local infrastructure.

Given that the testbed covers such a large region with different environments, this testbed provides a rich environment to study many aspects of vehicular networks, such as routing, system and application design and power management.

2.5 Content Distribution Approaches

In most vehicular network projects, the driving force of effort and development has been related to safety applications, providing a safer environment for the users to drive in. However, as those applications are developed and tested, new interests arise, regarding entertainment applications, capable of delivering multimedia files, software updates, tourist information, among others.

In VANETs, the concept of Content Distribution fits the following scenario. The original information is uploaded to the Internet server. Vehicles passing by the infrastructure, an RSU

or other access point, can download the information when there is appropriate connectivity. Afterwards, the vehicles with this information can disseminate it to nearby vehicles they come into contact with. In this manner, content is distributed throughout the entire network.

To overcome the short contact duration with the infrastructure, peer-to-peer technology can help due to its native characteristic of partitioning the information and distributing it amongst peers through file swarming [26]. Mario Gerla *et al.* [27] conducted a survey on content distribution schemes that have proven useful in vehicular networks and the following subsections will describe their characteristics, in order to determine which strategy could prove most useful in decreasing a vehicular network's overhead.

2.5.1 Peer-to-Peer Schemes

SPAWN

Swarming Protocol For Vehicular Ad-Hoc Wireless Networks (SPAWN), follows the same structure as a Peer-to-Peer (P2P) swarming protocol. Peers download pieces of a file from the infrastructure and then share the pieces amongst themselves. Figure 2.8 illustrates how a vehicle obtains content in the SPAWN strategy.

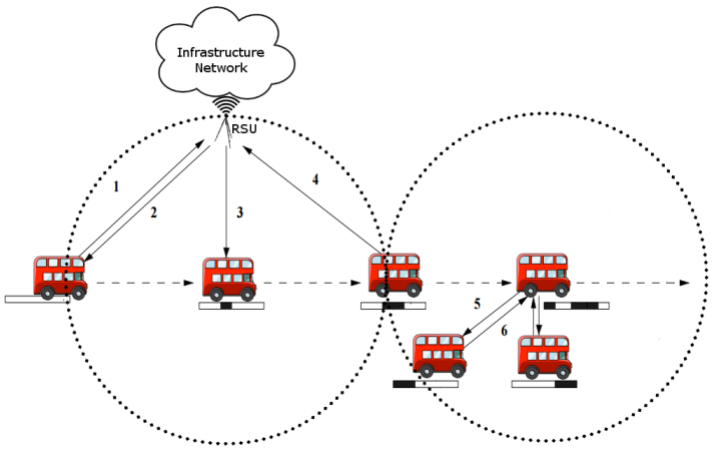


Figure 2.8: SPAWN Scheme, based on [28].

In step 1, the vehicle has entered the range of the RSU. In step 2, the OBU receives data from the RSU. In step 3, the vehicle is still within range and, as such, receives more content. In step 4, the vehicle is out of range of the infrastructure. In steps 5 and 6, the OBU is within range of other OBUs and start exchanging pieces of the downloaded file. Preliminary results of this scheme tested a random packet distribution whenever a car comes into range of a neighbour. Although SPAWN uses additional packets to control whether or not information should be sent, those were used for location-awareness of vehicles interested in receiving the packets, a type of strategy that has already been implemented extensively in several projects.

CarTorrent

According to [29], CarTorrent is another content distribution scheme similar to P2P file swarming which can be seen as an improvement to the SPAWN Protocol. This scheme incorporates Ad-Hoc On-Demand Distance Vector (AODV) routing to aid the vehicles in performing peer discovery and multi-hop message transmission, as well as allowing the vehicles to collect information about the file pieces. In this manner, in CarTorrent, the vehicles are capable of determining which is the rarest file piece in the network and the closest node that holds it in storage. This is yet another example of a location-awareness strategy.

RoadCast

In [30], a P2P content distribution scheme is proposed, based on increasing chances of obtaining the most popular content. By having nodes announce how popular certain content is, RoadCast ensures that the most popular content is delivered with higher probability. This method could prove useful in ensuring dissemination in slower experiments, as file popularity becomes more evident over time.

MDDV

Hao Wu *et. al.* [31] proposed a Mobility-Centric Data Dissemination (MDDV) algorithm based on geographical positioning. However, this algorithm is primarily used for V2V communications, that is, only OBUs would be considered. Moreover, since this algorithm sends packets based on geographical position of the OBUs rather than their content, it goes against the intended aim of this Dissertation.

REDEC

REceiver-based solution with video transmission DECoupled (REDEC) is a content distribution scheme thought specifically for video streaming. Its algorithm considers that a vehicle can be in one of four different states: non-relay, scheduled, relay or scheduled relay. A non-relay node does not broadcast packets and is only potentially interested in receiving them. A relay node actively broadcasts packets and receives them. A scheduled/scheduled relay node has received a control packet and is deciding if it will broadcast data packets or not. However, a scheduled relay node will forward packets even while in that decision process.

This category selection is meant to decrease network overhead by limiting the number of nodes capable of sending control packets. However, the percentage of nodes actively participating in controlling packet broadcasting, when data is to be disseminated, is always inferior to 20%. Given that the aim of this Dissertation is not to limit already-implemented strategies but to attempt to improve them, control packets are a key component of the vehicular network of study. As a result, this is not a preferred content distribution strategy.

PYRAMID

In PYRAMID, the OBUs resort to determining relevant content in neighboring vehicles by using probabilistic data structures to get an approximation of the neighbours' content [32]. The authors developed two mechanisms in order to do so: Task Prioritization and Content Reconciliation. The first one aims to identify the best vehicle to establish communication with and the second one uses a membership test to identify redundant content.

In this manner, the authors use two sets of "granularity" to approximate a node's desired content, Coarse Granularity for determining a neighbor's content through a probabilistic data structure and Fine Granularity to identify redundant information, as shown in Figure 2.9.

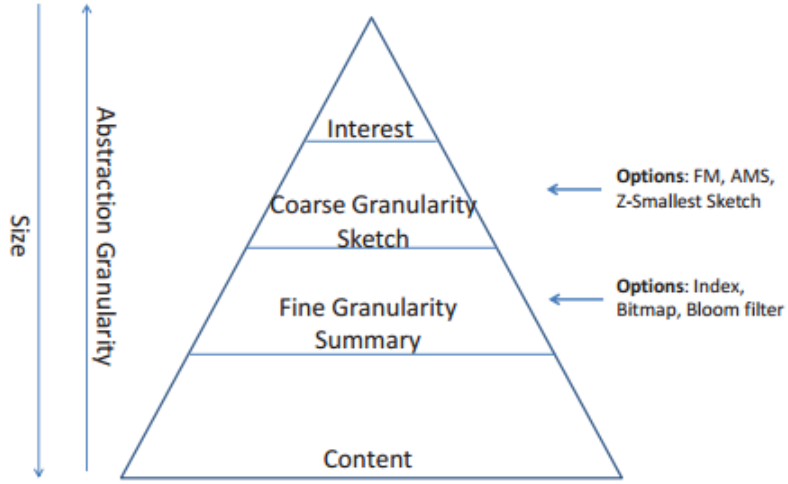


Figure 2.9: PYRAMID Abstraction of Contents [32].

The coarse granularity involves summarizing a neighbor's content by announcing to a neighbor which packets would be more useful to him. Although some risk in such a strategy would be expected, such risk is greatly decreased thanks to the fine granularity. The identification of redundant packets was tested with several strategies, one of which involves using a Bloom Filter, a data structure that is able to identify redundant packets with certainty.

The use of a structure to summarize content could be used to announce the contents of a vehicle to its neighbours, possibly improving the network's overhead. As such, the following subsection will provide the reader with a survey on content distribution strategies that have used Bloom Filters.

2.5.2 Bloom Filter Schemes

Bloom Filters are not a recent data structure and, even in the field of Vehicular Communications, they are not a novelty, either. The work in [33] has performed an extensive study on the use of Bloom Filters in Ad-Hoc Networks and the following subsections will describe their characteristics.

B-SUB

Yaxiong Zhao *et al.* proposed establishing Human Network (HUNET)s through wireless devices, such as smartphones, carried by moving users. Through this network, users are able to forward specific messages to other users, according to their interests. In this manner, devices don't perform any addressing or routing tasks but merely forward the message to brokers, responsible for content matching [34]. Bloom Filters are used to encode users' interests, consuming far less memory than other methods. Although it is not a vehicular network, this principle could also be applied in it given that HUNETs are composed of moving users. Figure 2.10 illustrates an example of a human network.

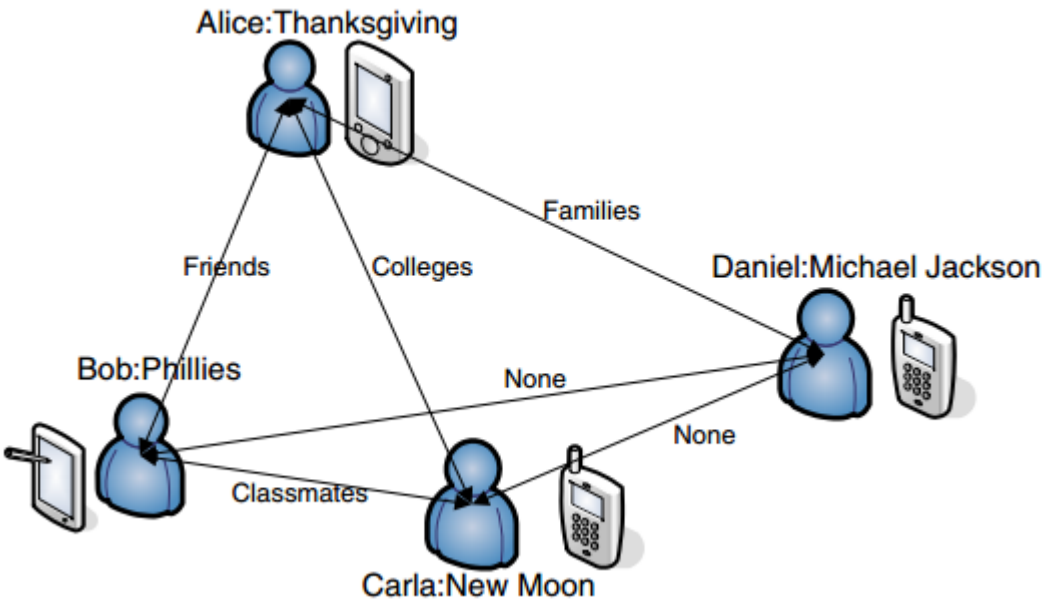


Figure 2.10: Example of a HUNET [34].

Extended ODMRP

On-Demand Multicast Routing Protocol (ODMRP) is a protocol for routing multicast and unicast traffic throughout Ad-Hoc Networks [35]. In [36], an extended version of ODMRP is suggested for data dissemination in content-based subscriptions. The authors proposed a Bloom Filter-based event dissemination system where messages are aggregated and summarized into a Bloom Filter and later sent to the most appropriate multicast group. Given the constraints of mobile networks, it is important in a real life network to send packets as quickly as possible. Through this manner, this could also be applicable to the control messages of a vehicular network, consequently decreasing the size of the control messages required for posterior data dissemination.

OSNR

Message Forwarding can be done in a network by taking advantage of other people's mobile devices. Iain Parris *et al.* [37] noted that performing this Social Network Routing enables a user to send messages via other users' smartphones and can therefore broadcast these networks, introducing privacy concerns. As such, an Obfuscated Social Network Routing (OSNR) protocol was designed, which uses a Bloom Filter to protect a user's friend list by encoding it rather than leaving it in plain text. This further strengthens the use of a Bloom Filter in a vehicular network. Not only does the encoding provide a smaller control packet size, it also hides its information, which could prove useful if sensitive data were to be exchanged between nodes.

BLooGo

BLOOm filter based GOssip algorithm for wireless NDN is an algorithm that allows nodes to send and receive messages without any prior knowledge of the network [38]. A node can decide whether to forward packets or not based solely on information about neighboring nodes encoded in a packet as a Bloom Filter. If a receiver of this information has all of its neighbours with the same content, the receiver knows it should not forward messages. Likewise, if a neighbor does not contain such information, messages are to be sent. Figure 2.11 illustrates this algorithm. This can be, potentially, the best approximation of this Dissertation's study. Through the use of a Bloom Filter, control messages become shorter and nodes will still be able to send content based on this approximate data.

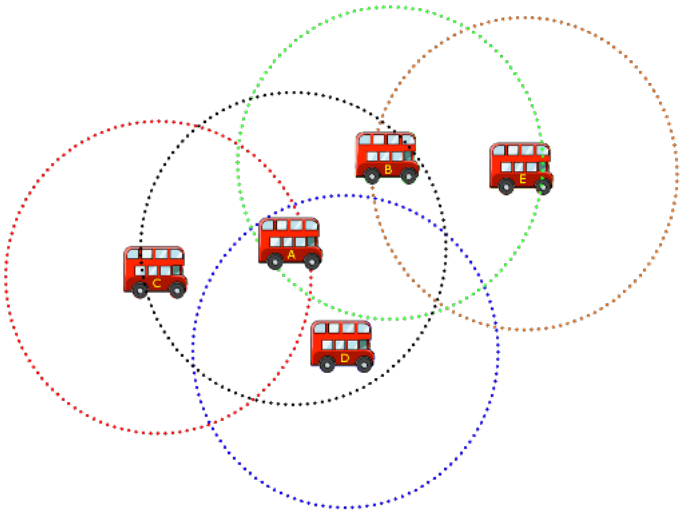


Figure 2.11: BlooGo Scheme, based on [38].

When vehicle A requests content, vehicles B, C and D will only forward that request to neighboring nodes if these vehicles do not contain such information. If either B, C or D have the requested information, they simply send it to A. If not, only B should forward A's request since it is the only vehicle that can reach a node that is not within A's range.

2.6 Vehicular Networks' Simulators

Before implementing new routing algorithms in a real-life vehicular network, those must be developed and tested in platforms that approximate reality as closely as possible. In this section, several simulators for general networks will be described and, furthermore, VANET and DTN simulators, built specifically for vehicular networks, with the aid of extensive vehicular network studies performed by [18] and [39], in order to emphasize that the number of solutions for network simulation are plentiful.

2.6.1 General Simulators

NS-2/NS-3

The Network Simulator (NS) NS-2 first appeared in 1989 as a network simulator with significant transport, routing and multicast capabilities. However, although it was able to simulate multiple radio interfaces, its wireless channel models were unrealistic and radio propagation was biased. Moreover, it only supported bi-directional or omni-directional signal propagation. Due to these shortcomings, NS-2 proved to have limited scalability, consuming excessive CPU and memory and was, thus, discontinued.

As a replacement for NS-2, an improved simulator named NS-3 was developed. Just like

the previous version, this simulator is written in C++, albeit with much shorter coding. NS-3 started by having only a few hundred lines as opposed to NS-2's 300,000 lines. Furthermore, this simulator contains proper support to simulate larger networks.

GlomoSim/QualNet

Global **Mo**bile Information System **Sim**ulator was developed in California, using Parsec and Java and was mainly targeted for wireless network simulation. It had the distinct trait of separating the network into modules, each one running as a different process, hence reducing the CPU load and allowing for higher scalability. Unlike NS-2, GlomoSim supported a far more extensive number of nodes and was able to support several different protocols and wireless technologies.

After Parsec stopped working on freeware software, QualNet was developed as a commercial version of GlomoSim. It is a C++ written simulator and contains a powerful 3D visualization tool as well as its own Network Analyzer. It is capable of simulating different types of networks, including MANETs.

NetSim

In [40], NetSim is described as an object-oriented architectural simulator. Written in C#, this network simulator can form a variety of computer networks, whether they be sensor networks, cellular, LTE, military, among others. NetSim as a simulator, brings several benefits. Its core language provides developers with easier programming, since C# does not require memory clean-up; its architectural accuracy makes it so that each class object is seen as a physical component of a machine and each public method as a communication line and, finally, due to accuracy, it is a highly flexible simulator since changing a component in the network only requires creating a new object in the code and overriding the old object's methods.

Unfortunately, all these benefits come with a cost in the actual network simulation. The added methods, dynamic objects and garbage clean-up generate additional overhead, ergo, the same simulation achieves different results when compared to other simulators.

2.6.2 VANET Simulators

In order to perform an accurate simulation of a vehicular network, network simulators need to be coupled with mobility models to simulate node mobility. The following simulators are some examples of this collaboration.

VanetMobiSim

VanetMobiSim is a simulator that includes many road elements' behaviour in order to more closely simulate a real road traffic scenario, namely stop signs, traffic lights and pedestrians. Although it is capable of using several different algorithms to determine the most viable path between source and destination, the data generated by this simulator cannot be

inserted in a network simulator. As a result, VANET simulation can only be achieved by using this simulator along with another, making it somewhat impractical to work with.

VANETsim

According to [41], VANETsim was developed primarily for studying security and privacy concepts in VANETs. Programmed in Java, VANETsim contains a very powerful GUI, which provides the user with a map editor, allowing the user to visualize simulations in maps created from scratch or in an imported map of a real location, so as to study real-life vehicle behaviour as closely as possible. Moreover, it enables the deployment of infrastructure. However, since the main aimed use for this simulator is security rather than content distribution, it is not an appropriate development platform.

TraNs

Traffic and **N**etwork **S**imulator is a VANET simulator based in Java, which integrates network and mobility simulators, such as Simulation Urban MObility (SUMO) [42] and NS-2 to simulate a VANET's behaviour. This simulator can be used in two different ways, depending on the need [43]. The network-centric method can be used to evaluate communication protocols that do not influence node mobility, such as the dissemination of non-urgent information. Secondly, the application-centric method can be used to evaluate applications that influence node mobility, such as safety-related applications. Unfortunately, this simulator has not been maintained since 2008 and, as such, was not considered.

NCTUns

National **C**hao **T**ung **U**niversity **N**etwork **S**imulator is a C++ written simulator with a unique characteristic. Rather than using a traffic simulator and a network simulator in conjunction (much like TraNs), NCTUns couples both these simulators into a single module. It is also capable of simulating multiple wireless interfaces, supports the IEEE 802.11p and WAVE standards and implements directional, bi-directional and rotating signal propagation. However, while most network simulators allow multiple TCP/IP versions in the same simulator, NCTUns only allows for one single version. Furthermore, the networks are only scalable to a certain limit; NCTUns can only withstand simulations with up to 4096 nodes in one simulation. Despite the lack of scalability, over four thousand nodes is still a considerable network but, to prevent future mishaps due to this constraint, this simulator was not considered as a development platform.

2.6.3 DTN Simulators

Regardless of the VANET simulator used, in order to implement and analyze a routing protocol developed specifically for a delay-tolerant network, a network simulator with that scenario implemented needs to be used, ergo, a DTN simulator.

DTN Simulator

In [44], the proposed DTN simulator was developed to evaluate routing algorithms in delay-tolerant networks. It is a discrete event simulator, written in Java capable of storing-and-forwarding messages for long periods of time, while sustaining network disconnection and link failures. This simulator also includes in its code, information expected to exist in a real-life scenario, namely, a limited storage size for each vehicle and limited bandwidth and delay during communication links.

The ONE

The Opportunistic **N**etwork **E**nvironment is another Java-based simulator with the main objective of evaluating routing algorithms in DTN scenarios [45]. However, while most simulators focus only on routing simulation, The ONE comes with additional capabilities such as its own visualization and analysis modules, statistical reporting, interfaces for importing and exporting mobility traces, events and messages, energy consumption data, among others. Included within it are also six different routing protocols, three mobility models and application support.

Despite the availability of these DTN simulators, it was decided not to use any of them. The DTN emulator available in our group, mOVERS, is based on the same code that is running in the real RSU and OBU boards, and any extension to the emulator is an extension to the real equipments, just requiring a compilation of the changes to the boards. Moreover, this solution is capable of using real data as a mobility pattern to simulate moving vehicles, and the real connectivity between vehicles, during these 24h, is also available to be used in the emulator. Since a 24 hour dataset on the pattern of moving vehicles already existed and no other DTN emulator could properly mimic this behaviour, mOVERS was ultimately chosen as the development platform for this Dissertation.

2.7 Summary

This chapter has provided the reader with a range of general information regarding Vehicular Ad-Hoc Networks, based on the literature. On a general scale, the basic concept of VANET was presented, along with its characteristics, main challenges and potential applications. On a technical scale, the reader was familiarized with a VANET's equipment, communication protocol, architecture and scope of dissemination.

Furthermore, the concept of Delay-Tolerant Networks was also provided, as well as its

architecture and applications. Finally, joining these two concepts, the notion of VDTN was elucidated along with its wide range of benefits and potential applications in the real world.

Moreover, a study on content distribution approaches was also provided with particular attention to projects which used Bloom Filters, a data structure which has proven useful in several projects, due to its coding and summarizing capabilities. Following that, a study on Network, VANET and DTN Simulators was presented, not only enlightening on the existence of many different types of simulators, each one with its own characteristics, but also enhancing the need for a simulator that can actively reproduce the unique traits of a DTN scenario, in order to study a proper content distribution scheme for said scenario.

The next chapter will discuss the DTN emulator used in the scope of this Dissertation. Despite the wide array of already-available simulators, mOVERS, a DTN emulator developed in a joint effort by IT and Veniam®, was used. This emulator will be described in the following chapter.

Chapter 3

mOVERS Module Description

3.1 Chapter Description

The main goal of this Dissertation is to develop a content dissemination strategy with minimum overhead possible using DTN communication in a VANET, ergo, in a Vehicular Delay-Tolerant Network. In order to do so, it is important to study the platform where a proper solution will be integrated and tests will be performed.

The chosen platform is called mOVERS, also known as mobile Opportunistic Vehicular Emulator for Real Scenarios. This chapter will describe the different modules of this emulator in detail, as well as the already-implemented content distribution strategies.
The organization of this chapter is as follows.

Section 3.2 presents mOVERS, giving a general description as well as delving into each of its modules, describing their function in the experiments' emulation.

Section 3.3 discusses the already implemented dissemination strategies, along with their advantages and disadvantages.

Section 3.4 summarizes this full chapter.

3.2 mOVERS

A partnership between IT and Veniam® started the development of a new implementation of DTN software, designed to operate in the VANET located in the city of Oporto, Portugal. It was developed in C/C++ programming language and supports vehicle communications, using the WAVE and 802.11p protocols. It is also capable of using the Wi-Fi and IEEE 802.11a/b/g protocols.

mOVERS is built using the same code in each emulated vehicle as the one in the real boards in OBUs and RSUs. It considers a modular architecture, as observed in Figure 3.1. It is composed by six main modules, each one with its own unique tasks: API Management, Communication, Logging, Neighboring, Storage and Routing. Modules work with each other through IPC sockets and nodes trade information with others through UDP ports. The following subsections will describe each module separately, for further clarity.

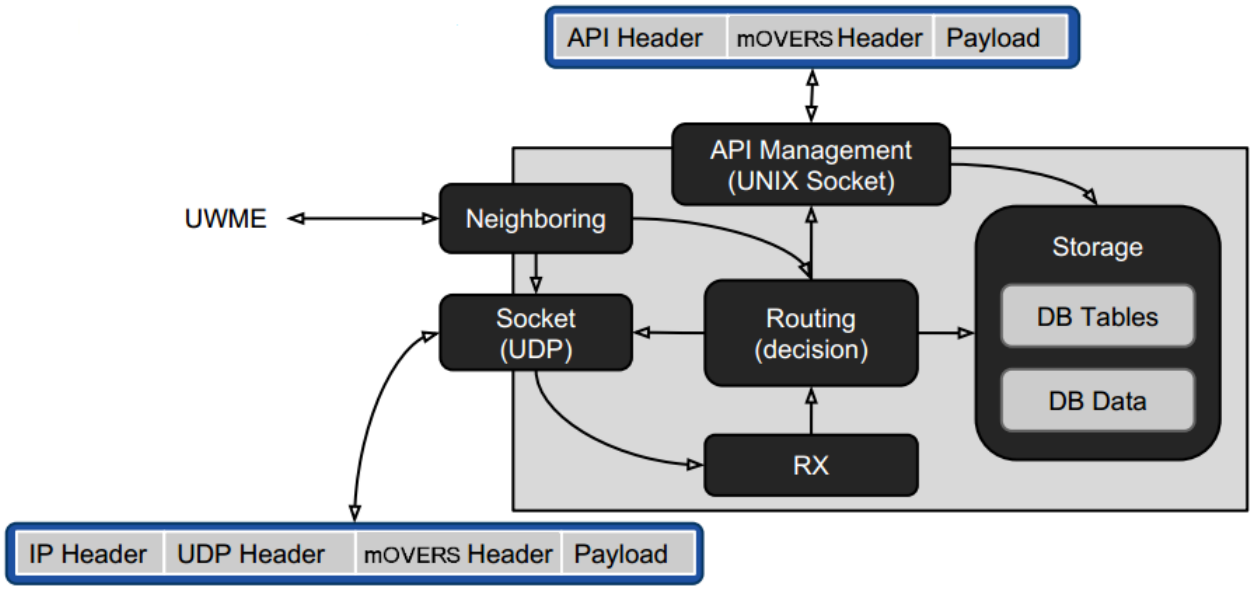


Figure 3.1: mOVERS Architecture [46].

3.2.1 API Management Module

Through this module, mOVERS nodes are capable of interacting with external applications. For testing purposes, some APIs are included in mOVERS, namely mOVERSPing, mOVERSSendString, mOVERSRecvString, mOVERSSendFile, mOVERSInBox, mOVERSMonitor and mOVERSCollectMonitor. To separate the Application module interactions from the rest of the network's elements, this module creates an abstraction layer to send or receive mOVERS packets to/from the API.

3.2.2 Communication Module

Both the RX and Socket modules are within the Communication module, since they both work together to process data exchanged between nodes. The RX module is constantly checking if the UDP socket has received any data to forward to a node. When data is in the socket, the RX module is responsible for analyzing the data's flags and forwards the packet to the Neighboring module or to the Routing module, depending on whether the information is a neighboring control packet or a data packet. If the UDP socket contains a neighboring control packet, the Neighboring module will then be responsible for using it to update information related to the node's neighbours. If it contains a data packet, the RX module will identify the packet as an ACK (Acknowledgement), ADV (Advertisement) or Data packet, and forward it to the Routing module for further handling.

The Socket module manages the access to the UDP socket and acts as the abstraction layer to send/receive packets between neighbor nodes.

3.2.3 Logging Module

This module is built within the Routing module, since all its methods are called by the Routing handling methods. However, its importance in content distribution simulation highlights it and detaches it as a unique module, as well. The Logging module is responsible for keeping a record of data obtained through the experiments. During a mOVERS experiment, a log file is generated per machine emulated (OBUs and RSUs) and is updated per timestamp with relevant information. The variables of relevance stored for further analysis are as follows:

- node_eid: Identifier of the node where information was generated;
- timestamp: UNIX timestamp of the registered entry;
- packets_stored_total: Total number of packets stored by a node since the beginning of the emulation experiment;
- packets_transmitted_total: Total number of files transmitted by a node since the beginning of the emulation experiment;
- packets_stored_per_timestamp: Number of packets stored every two seconds;
- packets_transmitted_per_timestamp: Number of packets transmitted every two seconds;
- packets_listened_per_timestamp: Number of packets heard in the wireless medium every two seconds;
- packets_recv_good_per_timestamp: Number of new packets stored by a node every two seconds;
- packets_recv_bad_per_timestamp: number of already known packets stored by a node every two seconds;
- control_packets_number_per_timestamp: number of advertisement packets transmitted every two seconds;
- control_packets_size_per_timestamp: Total size of advertisement packets transmitted every two seconds.

Once a mOVERS simulation ends, the Logging module returns, as an end result, a CSV file with all the above variables' values per timestamp to be posteriorly treated in MATLAB for statistical analysis.

3.2.4 Neighboring Module

This is the module responsible for discovering a node's neighbours. Depending on the interface used, neighbours are treated as four different types: Wi-Fi, WAVE, Ethernet and Static. The former three are neighbours which use that specific communication protocol and the latter applies to a neighbour with a predefined static route. In a VANET, typically, WAVE neighbours are the OBUs and RSUs, Wi-Fi neighbours are sensors, and Ethernet/Static neighbours stand for the RSUs in the core server in the infrastructure network. Figure 3.2 illustrates the four types of neighbours that this module is capable of handling.

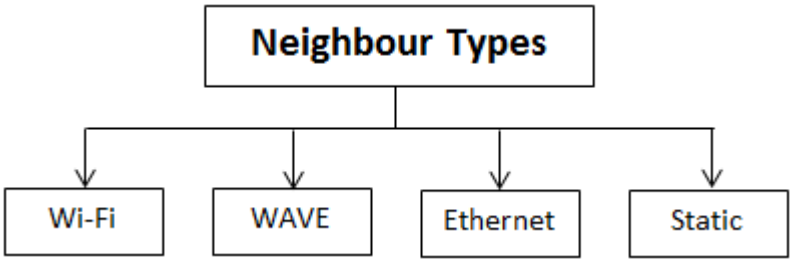


Figure 3.2: Neighbour Types

Throughout an emulation, this module is constantly searching for new neighbours. Whenever a node announces its presence to its neighbours, those update their internal structures with information regarding the new neighbour, namely its IP address, Endpoint Identifier (EID), type (OBU or RSU), communication port and RSSI.

3.2.5 Storage Module

This module is responsible for the nodes' packet storing, that is, for holding the data packets along with information required to forward them to a neighbour. Generally speaking, its most crucial methods involve pushing and pulling (storing and receiving) a packet from storage; peeking a packet to query for its destination EID, serviceID or expiration time; deleting from storage and querying for a specific packet's existence in storage. This module also contains two sub-modules, StorageRAM and StorageDisk. StorageRAM implements five in-memory information tables for fast querying, those being as follows:

- **Expiry Table:** Packets ordered by time remaining until expiration;
- **Own Table:** Packets ordered by serviceID, not meant for forwarding to other nodes;
- **NoData Table:** Packets ordered by expiry time, known in network but no data is stored;
- **Hash Table:** Packets ordered by its identifier.

StorageDisk has several functions: manages the packet storage on disk, performs read and write operations to obtain packets and uses the StorageRAM tables to optimize its performance. Figure 3.3 illustrates this module's organization.

Packets to be forwarded are always organized in the first table, ensuring that the packet with the lowest lifetime is always picked, preventing packet starvation.

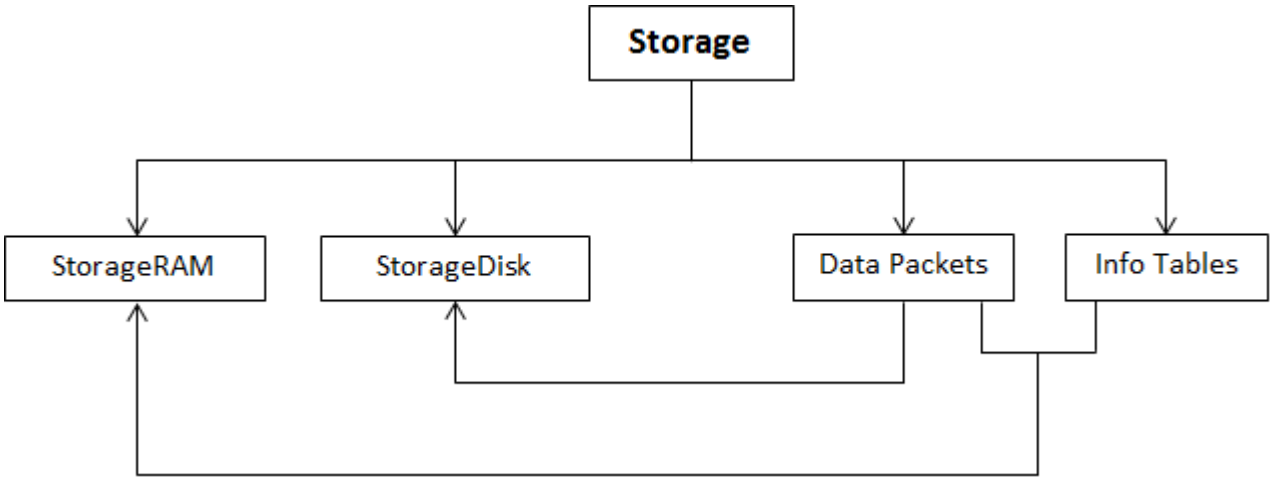


Figure 3.3: Storage Module Organization, based on [46].

3.2.6 Routing Module

The Routing module is the core module, responsible for handling received packets, deciding which packets to forward and to which neighbour they should be sent to. Once the RX module determines which type a packet belongs to (ACK, ADV or Data), it must be sent to an appropriate neighbour. mOVERS uses a hybrid solution by routing packets per Neighbour and per Packet: in order to send a packet, a node checks for neighbours in the area, chooses one and selects the packets that should be sent to this neighbour. In order to optimize the "Routing per Neighbour" operations, four neighbour types were considered when a node needs to select one, as observed in Figure 3.4.

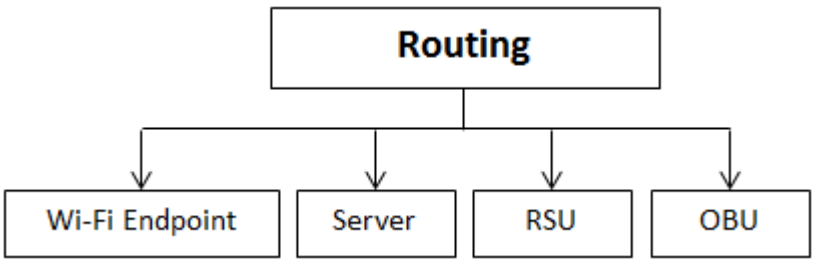


Figure 3.4: Routing per Neighbour, based on [46].

Since, for all simulations performed in the scope of this Dissertation, only OBUs and RSUs were considered, Wi-Fi Endpoints and Servers will not be considered. The most important methods in this module for treating information are the **ACKHandler**, responsible for handling acknowledgement packets; the **ADVHandler**, for treating advertisement packets and, finally, the **PKTHandler**, for handling data packets.

This module also contains two sub-modules named RoutingRSU and RoutingOBU and are used to handle sent and received packets. These two sub-modules include the above-mentioned methods, the ACK, ADV and packet handlers and invoke each one, according to the packet being handled.

3.3 Current Distribution Strategies

In order to select the right data to be sent and therefore, achieve a high delivery ratio, a content distribution strategy to select the right data to forward to a node's neighbours is a suitable approach. Some Content Distribution strategies have already been implemented prior to this Dissertation. These strategies are named LNHF (Least Number of Hops First), LRBF (Local Rarest Bundle First) and LRGF (Local Rarest Generation First). The following sub-sections will explain each strategy in a broad scope but can be studied in further detail in [16].

3.3.1 Least Number of Hops First

This strategy aims to order a node's packets by the number of nodes it has traveled. Due to the Store-Carry-and-Forward mechanism, when a node receives a data packet, it stores it, carries it and forwards it once neighbours are within the vehicle's range. Once a neighbour receives a packet, it updates its internal information by incrementing the packet's number of hops. As a result, this content distribution approach is based on directly associating a packet's number of hops with the number of neighbours that contain this packet. The higher the number of hops of a packet, the higher the likelihood of this specific packet being stored in other nodes.

On another hand, since a packet's number of hops in an RSU is always zero, a second parameter is also added; the number of transmissions. Since the sender node creates a copy of the packet to forward, this node can count the number of copies it creates and, by extension, the number of transmissions. The higher the number of copies, the higher the likelihood of this specific packet being stored in other nodes.

As a result, LNHF is a content distribution strategy that selects a packet based on the number of hops in OBUs and on the number of transmissions in RSUs. Figure 3.5 summarizes the nodes' behaviour when sending or receiving a packet with this strategy.

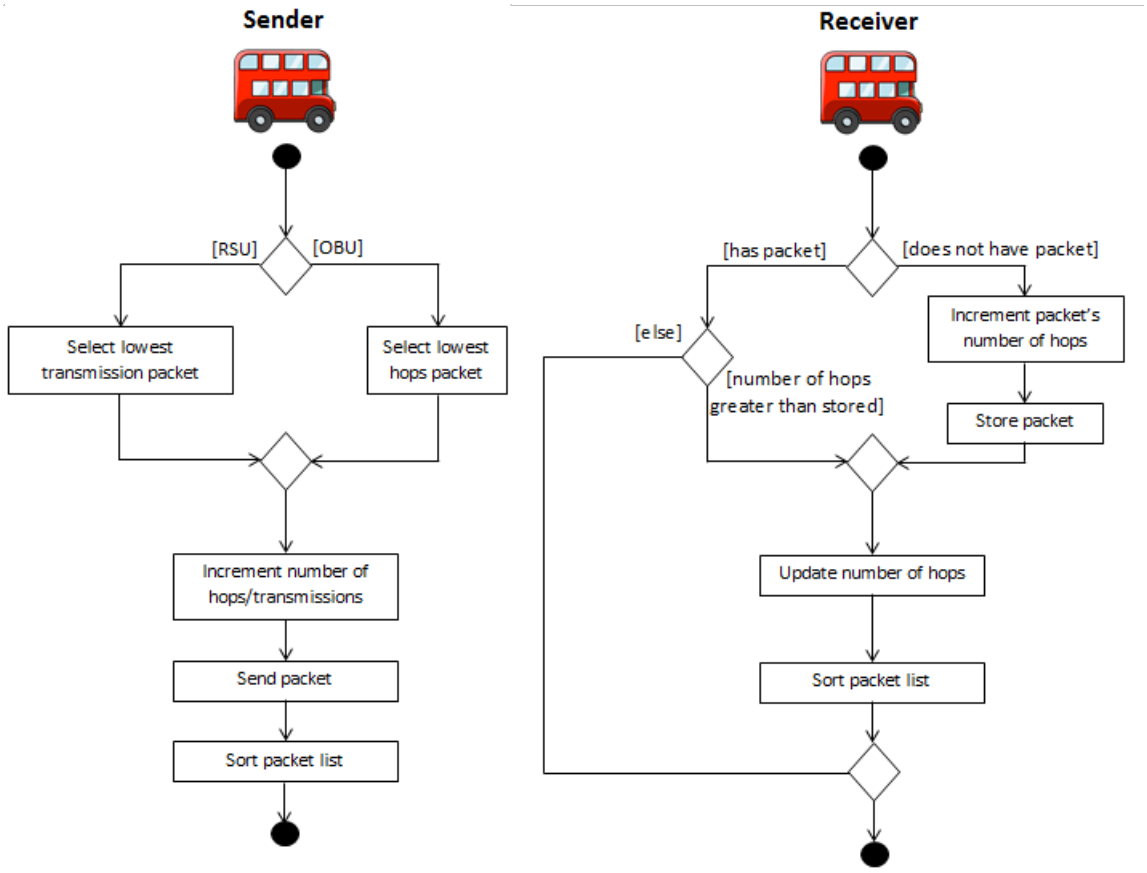


Figure 3.5: Least Number of Hops Strategy, based on [16].

During transmission, when a node finds a neighbour, if the node itself is an OBU, it will find the packet with the lowest number of hops and, if it is an RSU, lowest number of transmissions. After incrementing that variable, the packet is forwarded and the list is sorted. During reception, if the receiver node has the packet, it will update that packet's number of hops if the received number is higher. However, if it does not contain the packet, it increments the number of hops in the received packet, stores it and the list is sorted.

Tests performed with this strategy have proven that a slight increase in overhead is present in the network due to the addition of two new fields to order the packets to be sent. Moreover, due to the lack of control packets, the amount of data packets that the network transmits is more constant, as opposed to other strategies which limit the transmission decision. However, the delivery ratio is smaller.

3.3.2 Local Rarest Bundle First

The previous strategy focused on implementing the minimum control data possible so as not to significantly increase the network overhead. This strategy, on the other hand, focuses

on using a control packet to influence the decision on which packets to be forwarded next by implementing knowledge on the neighbours' storage content. The design is similar to torrent schemes since it finds the rarest piece, the rarest packet amongst neighboring nodes and forwards it. As such, every time a node is to send packets, it has knowledge on each neighbour's storage. It is in this strategy that Advertisement Packets are used.

Advertisement packets are a type of control packets used in this strategy to share information regarding a node's storage content, specifically it shares the identifier of the node, its files and the packets (pieces) of each file that the node contains. Figure 3.6 showcases this strategy's Advertisement Packet structure.

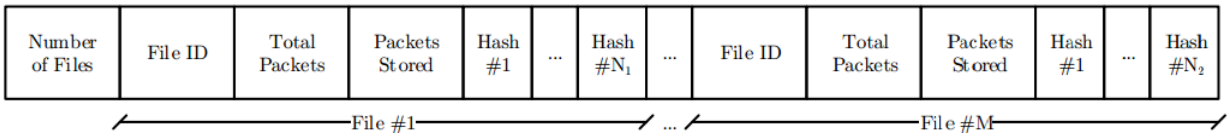


Figure 3.6: LRBF's Advertisement Packet Structure [16].

As a result, LRBF is a content distribution strategy based on selecting the rarest packet among neighboring nodes by analyzing the neighbours' storage content, information received through a control packet sent by the neighbours themselves. Figure 3.7 summarizes the nodes' behaviour when sending or receiving an advertisement packet with this strategy.

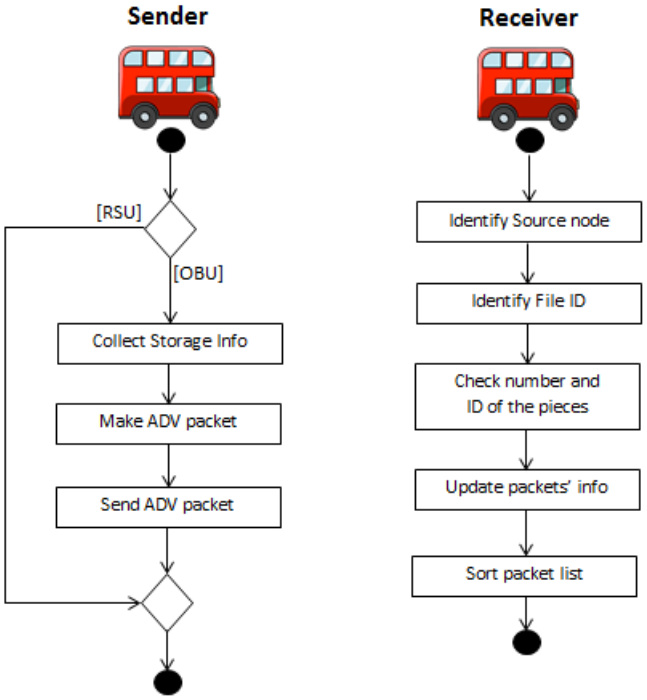


Figure 3.7: Local Rarest Bundle First Strategy - ADV Packets, based on [16].

During transmission of an advertisement packet, the sender node gathers information about his own storage, specifically, file identifiers, total number of packets that compose each file and the number of packets that the node contains per file. After collecting this information, the packet is built and sent in broadcast to all neighbours at that point in time. During reception, the receiver node identifies the content stored by that specific sender and updates the number of nodes associated to each packet in an internal structure. Afterwards, the packet list is re-sorted to ensure that the most lacking packet is the first one to be peeked during data transmission.

Once an Advertisement packet has been received, the receiver node is able to transmit data packets with knowledge of which packet is most needed by its neighbours. Figure 3.8 summarizes the nodes' behaviour when sending or receiving data packets with this strategy.

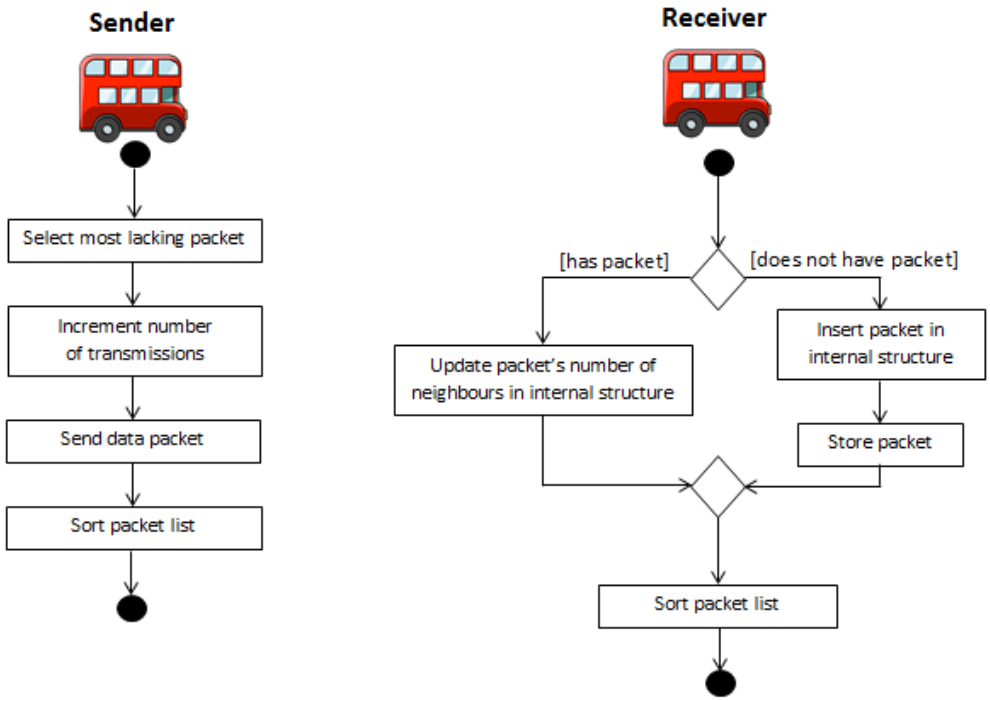


Figure 3.8: Local Rarest Bundle First Strategy - Data Packets, based on [16].

During transmission, as soon as a neighbour enters the sender node's range, it will select the most lacking packet (information derived from advertisement packets), increment the number of times it was transmitted and send it. Afterwards, it will sort its internal structures, since a new node contains what was then the most lacking packet. During reception, if the receiver node already contains the packet it received, the number of neighbours that contain this packet is updated and the internal structure is re-sorted. However, if it does not contain the packet, the packet is added to the internal structure responsible for monitoring the number of neighbours. The packet is then stored in Storage and the list is re-sorted.

When compared to LNHF, this strategy is costly in terms of network resources consumption and, by extension, network overhead due to the introduction of an additional packet to be forwarded, prior to data packet forwarding. Although previous testing has proven that this additional information leads to a higher delivery rate, it also introduces a considerable network overhead, which increases as the size of the advertisement packet increases. Moreover, the higher the frequency at which these packets are sent, the higher the network overhead as well. On the other hand, packet transmission decreases overtime since the advertisement packets limit transmission.

3.3.3 Local Rarest Generation First

This strategy is similar to the previous one except that, rather than focusing on a single "rarest packet", this approach focuses on finding the "rarest group of packets" while, at the same time, it implements network coding to reduce rebroadcast. This was achieved by splitting a file into several blocks and assuming Random Linear Network Coding (RLNC) in the packets. In this manner, packets are identified as belonging to the same generation if they belong to the same block. Furthermore, upon packet reception, packets can only be decoded if enough packets of the same generation comprise a block. Intermediary nodes are only capable of forwarding coded packets that were coded by other nodes and can only generate its own coded packets once it has decoded a complete block of packets. This strategy also requires an internal structure, this time to associate the collected information with the generations stored, in order to identify if a generation (a file) has been completely received or not.

This methodology was used with the aim of decreasing the network overhead because, although advertisement packets are still present in this strategy, they no longer advertise all storage content of a node but rather information regarding content blocks and number of coded packets per block (rank). Moreover, with network coding, sending specific packets is not required to recover the original decoded packets, but merely a number of packets in the same generation. Figure 3.9 showcases this strategy's Advertisement Packet structure.

Figure 3.9: LRGF's Advertisement Packet Structure [16].

As a result, LRGF is a content distribution strategy based on selecting the rarest group of packets among neighboring nodes by analyzing the neighbours' blocks of coded packets. Figure 3.10 summarizes the nodes' behaviour when sending or receiving an advertisement packet with this strategy.

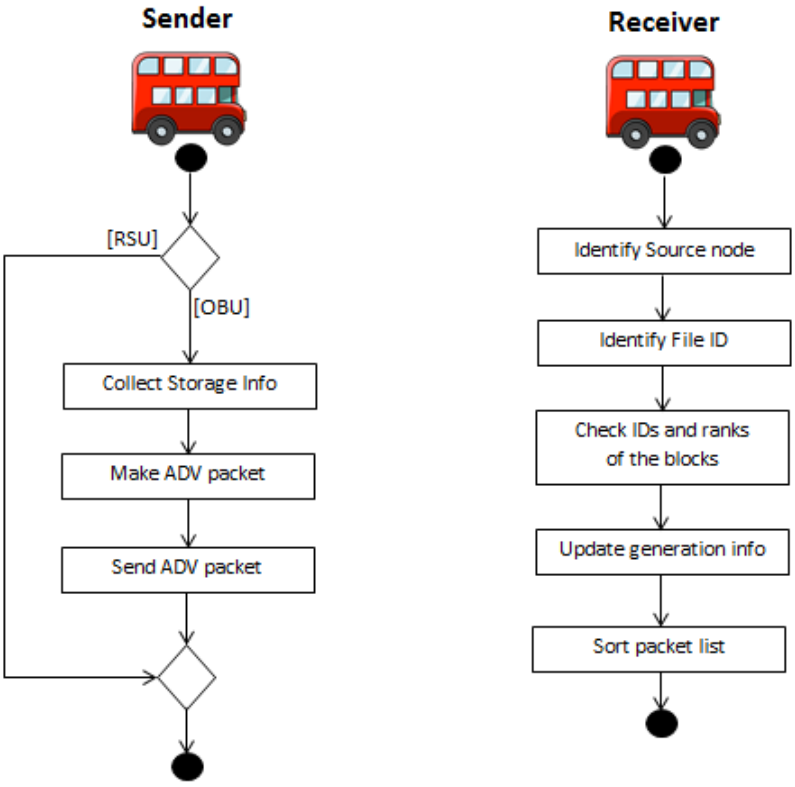


Figure 3.10: Local Rarest Generation First Strategy - ADV Packets, based on [16].

During transmission of an advertisement packet in LRGF, much like LRBF, the sender node collects information regarding its own storage content. However, this time, it will collect file identifiers and, for each file, the number of blocks and number of coded packets within each block, thereby potentially reducing the size of the advertisement packet. Once all this information is collected, the advertisement packet is created and sent in broadcast to all neighbours within the sender node's range. During reception, after identifying the sender node and each file, it will identify each block and rank and update the information it contains on each file. Afterwards, the storage is sorted to ensure that the first packets to be sent are part of the most lacking generation.

Once an Advertisement packet has been received, the receiver node is able to transmit the coded packets it contains with knowledge of which packets are most needed by its neighbours. All of this while ensuring that the advertisement packets do not transmit as much information as when compared to the LRBF strategy. Figure 3.11 summarizes the nodes' behaviour when sending or receiving data packets with this strategy.

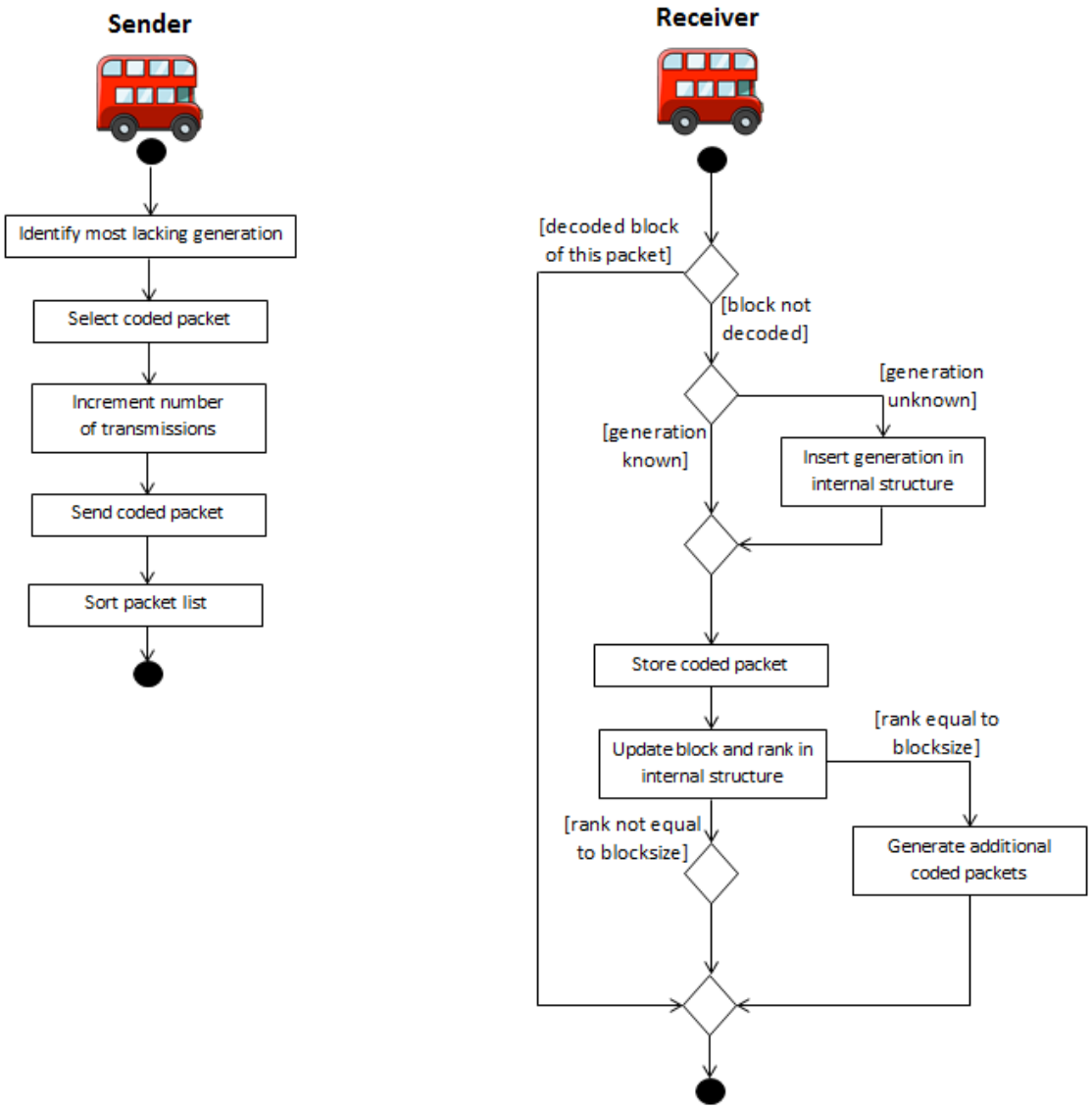


Figure 3.11: Local Rarest Generation First Strategy - Data Packets, based on [16].

During transmission, when the sender node has at least one neighbour lacking a coded packet from a generation that the node has, it will identify the most lacking generation (information derived from advertisement packets) and select one of the coded packets at random. Then, it will increase the number of transmissions of that coded packet and send it in broadcast. Since a packet was transmitted, the internal structures are updated. During reception, if the receiver node has decoded the block associated with the received packet, nothing is done. If the block is not decoded and there is no entry of this packet in the internal structures, the packet's generation is added to it. Afterwards, the coded packet is stored and the block and rank associated with it are updated. Once a node has collected a number of coded packets that equal the block's size, additional coded packets will be generated to emulate coding.

3.4 Packet Structure

The mOVERS implementation does not strictly follow the specifications of the Bundle Layer. Therefore, for consistency purposes, the packets used in this development platform will be referred to as mOVERS packets. The structure of a mOVERS packet is as shown below:

- **mOVERSHeader**
  - *Version:* The emulator version;
  - *ServiceID:* Neighbour Discovery or Content Distribution service;
  - *Source and Destination EIDs:* Identifying a packet's end-to-end path;
  - *Destination Information:* Node type, for example;
  - *Previous EID:* The previous holder of the packet;
  - *Hash:* Packet's identifier;
  - *Expiry Date:* Time of creation and lifetime of the packet;
  - *Data Length:* Size of the packet's data;
  - *Options Length:* Variable options imposed by user (yet not added in emulator);
  - *Priority:*
  - *Number of Neighbours:* Number of nodes containing this packet;
  - *Flag:* Identifier of the packet's type;
  - *File Identifier:* Used with previously implemented content distribution strategies. Identifies the file to which the packet corresponds to;
  - *Total Packets of File:* Used with previously implemented content distribution strategies. Identifies the number of packets that compose the file;
  - *Block Identifier:* Used with previously implemented content distribution strategies. Identifies the block to which the packet corresponds to;
  - *Block Size:* Used with previously implemented content distribution strategies. Indicates the size of the block;
  - *Generation Size:* Used with previously implemented content distribution strategies. Identifies the current rank of the associated block.
- **Data**

3.5 Emulation Procedures

Finally, now that all modules of the mOVERS emulator have been introduced and studied, it is also important to grasp how all the modules operate. mOVERS is a multi-thread software to avoid race hazards (when the program does not run in the order intended). Figure 3.12 illustrates the emulation procedure when running mOVERS.

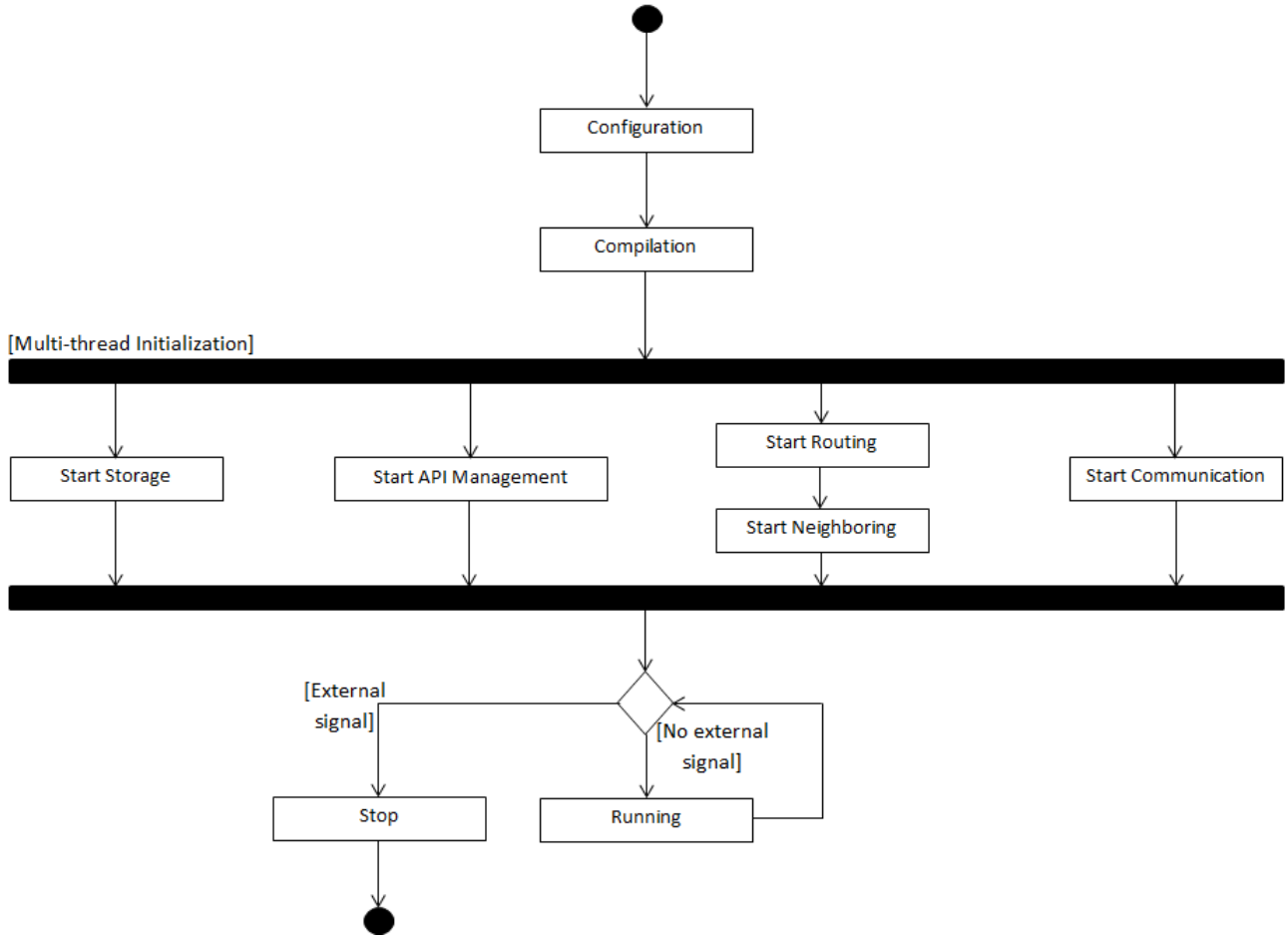


Figure 3.12: Emulation Procedures, based on [16].

Before launching all the threads of each module, a configuration file specifying socket port, communication interfaces, storage path and capacity, logging path and version of content distribution strategy is analyzed. Only after reading this file, the modules are initialized, launching a thread. Once all threads are launched, mOVERS runs until an external signal is sent.

3.6 Summary

Despite several VANET and DTN simulators exist, their use is not fully optimized for handling the unique traits of a vehicular network. As a result, a specialized solution has been developed for DTNs in VANETs, called mOVERS, mobile Opportunistic Vehicular Emulator for Real Scenarios was used.

Working as the development platform for this Dissertation, mOVERS is the DTN implementation that was used to develop the proposed content distribution strategy. The following chapter will clarify the problem that this Dissertation adresses along with the proposed content distribution strategy to solve it.

Chapter 4

Bloom Filter based Content Dissemination

4.1 Chapter Description

The main goal of this Dissertation is to develop a content dissemination strategy with minimum overhead possible using DTN communication in a VANET, that is, in a Vehicular Delay-Tolerant Network. In order to do so, understanding the inner workings of the development platform to be used was crucial to this Dissertation. Likewise, studying the already implemented dissemination strategies' advantages and disadvantages was required to grasp the possible improvements that could be added to the content dissemination mechanism.

This chapter provides insight on the problem to be addressed and the proposed solution for it, clarifying the solution's concept, variety of implementations and applications. Afterwards, it develops on the changes to the emulator and functions required to develop and integrate this solution.

The organization of this chapter is as follows.

Section 4.2 describes the problem that this work aims to solve, namely designing a content distribution strategy to decrease the network overhead.

Section 4.3 describes the proposed strategy to establish low congestion during a vehicular network experiment, the main concept, the different types of said strategy, generic routing process and its potential applications.

Section 4.4 describes the main modifications performed on the mOVERS emulator in order to implement the proposed content distribution strategy.

Section 4.5 summarizes this full chapter.

4.2 Problem Statement

Since VANETs and DTNs are characterized by intermittent connectivity and potentially long delays, certain mechanisms are mandatory to ensure safe data transfer. For DTN applications, the *Store-Carry-Forward* mechanism is used, having nodes storing the packets, carrying them and only transmitting them when the proper opportunity arises.

In VANETs, such a decision can be based on information about the nodes, much like previous content distribution schemes where packet information or neighbour information was disseminated before the data packet itself. This is called **Opportunistic Forwarding**, which consists of deciding which packet to send, based on information exchanged between vehicles about their storage content [47].

Depending on the algorithm, opportunistic forwarding can be faced with some challenges. For example, at the beginning of a VANET scenario, there may not be enough historical information to provide a decision on which packet to send. Other situation would be where the environment is so dynamic and in constant change that algorithms such as route prediction could not be used.

To surpass these scenarios' circumstances, **Epidemic Forwarding** can be a better alternative. This type of routing mechanism does not decide how to route the packet based on external information, such as vehicles' content or sensor data. Instead, packets are constantly being sent and will, therefore, eventually reach the destination [48]. The major issue of this routing mechanism is handling the redundant packets to avoid network overhead.

The already implemented content distribution strategies follow Opportunistic Forwarding and, to ensure the proposed strategy focuses on decreasing network overhead, the proposed solution will also follow such methodology. Therefore, the sender nodes must be able to determine which packets to forward based on information from neighbours.

Within this context, this Dissertation aims to, having prior knowledge of other content distribution schemes deployed in other projects, propose a content distribution strategy with the main objective of decreasing the network's overhead. As a result, this methodology follows Opportunistic Forwarding. It is worth emphasizing that DTN routing strategies have been previously tested in previous works and that the main target of this research is content dissemination, specifically, video dissemination among nodes.

4.3 Proposed Solution

Due to previous content distribution strategies relying heavily on control packets' information, the network can suffer from network overhead, not only due to the amount of advertisement packets being sent but also due to their size. A previous strategy implemented on mOVERS emulator, LRBF, could potentially result in nodes sending an exceedingly large advertisement packet, depending on the amount of packets that node contained in its storage.

A thorough survey of the literature on Content Distribution Schemes suggests that a Bloom Filter could be used to reduce the overhead associated with these advertisement packets. Although not many works state more than a few sentences about Bloom Filters, all of them seem to come to a consensus that Bloom Filters are a good alternative when it comes to reducing the size of an exchanged vector of values. As such, to decrease the network overhead, the proposed solution is a dissemination mechanism using a Bloom Filter. Moreover, a Bloom Filter offers space advantage when compared to other data structures (hash tables and binary trees, for example) since it does not store the actual packets, as well as time advantage, due to its fast membership querying.

4.3.1 Bloom Filter

A bloom filter is a space-efficient randomized data structure used to represent a set in order to test membership queries. Bloom filters can cause false positives, but the amount of space saved often outweigh this drawback when the probability of a false positive is sufficiently low.

This data structure dates back to 1970 for usage in database applications, but it has gained appeal in other areas such as networking [49].

Concept of Standard Bloom Filters

As previously mentioned, a Bloom Filter is a data structure used to represent a set and to test whether an element belongs to that set. It is similar to a hash table, except it does not store the actual values but rather hashes them in an array of bits. An example of a Bloom Filter can be seen below in Figure 4.1 along with a description of how a Bloom Filter works.

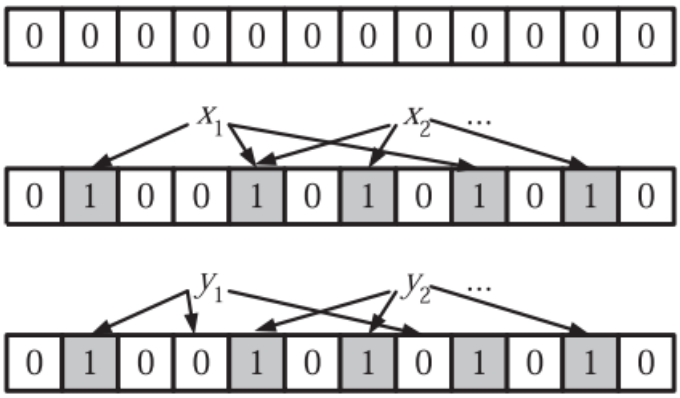


Figure 4.1: Bloom Filter example [49].

In mathematical terms, a Bloom Filter represents a set $S = \{x_1, x_2, ..., x_n\}$ of n elements in an array of m bits, all initialized at 0. For hashing the elements, the Bloom Filter also uses m independent hash functions to cover the bit array's range. Each element $x \in S$ is hashed m times with each hash indicating the location of one bit. That bit is then set to 1 in the array. To check if an item y belongs to S, that item is also hashed m times and the corresponding bits are checked. If all of those bits are set to 1 in the array, the element is either in the set or the filter generated a false positive. However, if even a single bit of hashed item y is not set to 1 in the array, the item cannot be in the set. In Figure 4.1, it can be observed that element y_1 cannot belong in the set since one of its bits is set to 0, while y_2 either belongs to the set or the filter has generated a false positive.

Provided that the probability of a false positive is small enough, the benefits outweigh the risks. Assuming that the hash functions are random, the false positive rate (p) for an array of size m, number of elements n and number of hash functions k of a Bloom Filter is described in the equation 4.1 [50].

$$p = \left(1 - \left(1 - \frac{1}{m}\right)^{kn}\right)^k \simeq \left(1 - e^{\frac{-kn}{m}}\right)^k = \left(1 - p\right)^k \tag{4.1}$$

Solving the equation for k, the correlation between the optimal number of hash functions with bit array size and number of elements to minimize false positive rate is given in the equation 4.2.

$$p = \left(\frac{1}{2}\right)^k = (0.6185)^{\frac{m}{n}} \tag{4.2}$$

Types of Bloom Filters

Since Bloom Filters date back to nearly 50 years, several changes have been made to the original concept, depending on the intended use. This section will describe the different types of bloom filters, and ultimately provide a brief summary of the benefits and drawbacks of each one [49].

Compressed Bloom Filters

When both sender and receiver have memory resources but the packet size is a critical factor during the transmission, a compressed bloom filter improves the performance of the standard bloom filter by increasing the decompressed storage size of a bloom filter while maintaining the bit transmission size constant [51].

While in the standard bloom filter, an optimal number of hash functions is calculated given the size of the array and number of elements to be inserted, this type of filter optimizes the number of hash functions given the size of the compressed data to be transmitted. As such, the new false positive rate equation is as shown in equation 4.3.

$$\phi^k = (0.6185)^{\frac{m}{n}} \tag{4.3}$$

where ϕ is the probability of each entry in the array to be set at 1.

Counting Bloom Filters

This improvement to the standard bloom filter involves the possibility of deleting an element from the bit array. Insertion of elements is a process of hashing the elements a given number of times and setting the resulting bits to 1. However, deletion is not as straightforward.

In a Counting Bloom Filter, each entry is not a bit but a counter. Therefore, whenever an

element is inserted, certain counters will be incremented. As a result, deletion is a matter of decreasing the resulting counters in the array. To avoid counter overflow, Fan *et. al.* [52] has proven that 4 bits per counter serves for most applications.

The probability that the *i*th counter overflows by reaching the value 16 in an array of size m is as given in equation 4.4.

$$P(max(c(i)) \geq 16) \leq 1.37 \times 10^{-15} \times m \tag{4.4}$$

In practice, when using a counting bloom filter, if an overflow occurs, an option is to leave the counter at its maximum level, generating a false negative if a counter reaches zero when it should not.

Space Code Bloom Filters

This bloom filter extends the standard's capabilities by allowing several sets of elements, enabling the bloom filter to count the number of occurences of an element, consuming fewer memory resources.

This type of bloom filter can be implemented by using several groups of hash functions. Insertion of an element begins by randomly selecting a group of hash functions and the resulting bits are set in the bit array. Querying for membership, however, requires two stages and therefore is more complex.

In the first stage for membership query, the number of hash groups matched by the element are selected. In the second stage, a likelihood estimation is calculated to estimate the number of times that an element is in the set. This results in more computing and bit size when compared to the standard bloom filter.

Attenuated Bloom Filters

This type of bloom filter uses a bi-dimensional array of bits where each row is a standard bloom filter. Element insertion involves using some form of control information. In [53], the number of hops was used to determine which row to insert the element in, for example. The element is then inserted in that row and all the other rows that follow.

The querying process is a simple iteration over the array, but the result is a tuple where the first value indicates whether the element may be present in the filter or not at all, and the second value indicates the several rows where the element is.

Dynamic Bloom Filters

In order to adapt bloom filters to even more scalable environments, this type creates a new bloom filter each time the false positive rate threshold of the previous filter is reached. Dynamic bloom filters require two new variables: s, the number of bloom filters in a matrix, and n_r, the number of elements inserted in a bloom filter.

Element insertion has two stages. Firstly, it checks which is the current bloom filter being used in the matrix. Secondly, it checks if the number of elements in that bloom filter has not reached its maximum capacity. Given these two operations, the element is then inserted.

Querying for membership is almost equal to the standard bloom filter. By checking which

bloom filter in the matrix has the bits of that element set to 1, if even one filter confirms this comparison, the element belongs in that matrix. Otherwise, it does not. These filters lose to other types in terms of array size and querying speed but, because filters can always be created, the growth is theoretically infinite.

Scope Decay Bloom Filters

This type of bloom filter detaches itself from all the others since this particular one can track time on when an element has been in the set. This filter has a probability associated with past membership of an element.

It is equal to the standard bloom filter, both in bit array structure to represent the set, group of hash functions and insertion of elements. It is only different in two ways: it contains a method to partially remove information from the bloom filter and a new method to query for membership. Partial removal involves instructing the filter to set some bits to 0, simulating a situation where an element once belonged to it. Query membership does not check if all resulting bits of an element are set to 1, but checks the number of bits set to 1, ranging from 0 (is not in the filter) to the number of k hash functions (probably is in the set at present).

Summary of Bloom Filters

The following table summarizes the types of bloom filters, as well as their benefits and drawbacks for easier comparison.

Type	**Advantages**	**Drawbacks**
Standard Bloom Filter	-	-
Compressed Bloom Filter	Smaller size (when compressed)	Bigger size (when decompressed)
Counting Bloom Filter	Deletion method Counting number of elements	Bigger size
Space Code Bloom Filter	Deletion method Counting number of elements	Higher false positive rate
Attenuated Bloom Filter	Weighted value upon query	Bigger size
Dynamic Bloom Filter	Scalable filter	Bigger size More operations
Scope Decay Bloom Filter	Time tracking	Existence of False Negative Rate

Table 4.1: Types of Bloom Filters

Since the emulator used has never had a bloom filter structure inserted in it, for algorithm testing purposes, we have chosen to implement the standard bloom filter, which offers a direct advantage, efficiency in approximating content using only bits.

Applications

According to [49], there are four types of network-related bloom filters:

- **Network Collaboration:** Bloom Filters can summarize content to facilitate node collaboration in networks. One example would be to check for similar content in moving nodes in a VANET;
- **Resource Routing:** Bloom Filters allow the use of probabilistic algorithms for locating resources. For example, web caching can be implemented with the aid of a Bloom Filter to significantly reduce the disk write workload, saving cache space on disk and increasing the cache hit rates [54];
- **Packet Routing:** Bloom Filters can simplify packet routing protocols, potentially speeding up the process;
- **Infrastructure Measurement:** Bloom Filters can provide a useful tool for analysis in the infrastructure related to, for example, sensor data.

The theme that unifies these applications is that a Bloom Filter offers a succinct way of representing a list of items. To separate the potential uses of Bloom Filters in four categories can be seen as a loose categorization, since some applications may fit in more than one category.

Dissemination Strategy using Bloom Filter

In order to perform a forwarding decision, nodes can use control information from other nodes, advertising some relevant data before sending the actual data packets. These control packets would normally contain one or more fields, depending on the dissemination strategy. However, it comes as a trade-off. The more control information being sent, the bigger the advertisement packet and the bigger network overhead.

Using a standard bloom filter, the only information this control packet will contain in its payload is the bit array with the hashed packets from the sender node. However, because the number of hash functions and inserted number of elements is required, the header of the packet will include that information.

The sender node will create the bloom filter by establishing the number of packets to be inserted, calculating the appropriate size of the bit table, number of hash functions and false positive probability. Once the sender node has created the bloom filter, it inserts its elements in the filter. Afterwards, the bit array is encapsulated into a buffer and sent in a packet with a header containing relevant information about the bloom filter. As a result, the advertisement

packet is constructed and then sent in broadcast to its neighbours. The Figure 4.2 shows the procedure for sending an advertisement packet.

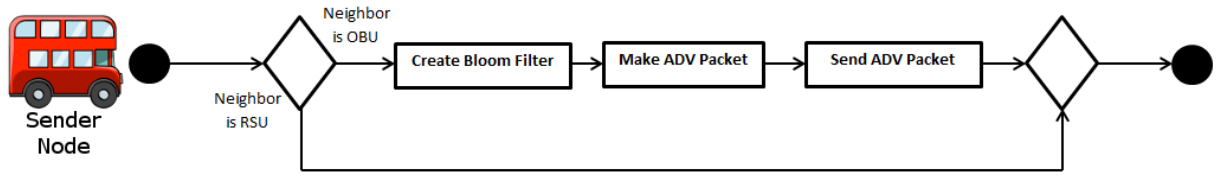


Figure 4.2: Procedure for sending an advertisement packet.

Once the receiver node has received the advertisement packet, it will decapsulate the packet, obtaining the bit array. However, the bit array by itself cannot be used to query elements for membership in a bloom filter. Therefore, the receiver node must also obtain the number of hash functions and number of elements in the bit array, thereby "re-assembling" the filter on the receiver node's side.

Once the receiver node has recreated the sender node's bloom filter, it will query its own packets for membership. If the receiver node contains at least one packet that is not present in the sender node's bloom filter, the receiver node can broadcast data packets. The Figure 4.3 shows the procedure for receiving an advertisement packet.

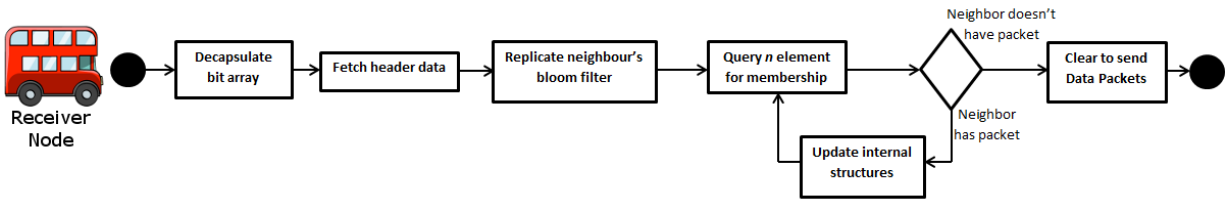


Figure 4.3: Procedure for receiving an advertisement packet.

4.3.2 Congestion Control

Congestion control is a major issue in vehicular networks due to their characteristics. Due to intermittent connectivity, packets can remain in a node's storage for extended periods of time, waiting for a transmission opportunity. As a result, upon contact, once a node understands that its neighbour requires packets, it must know which packets to send. However, to prevent nodes from never being sent, constantly being at the "bottom" of the storage, a control method is required.

Concept

To improve this fairness in a VANET scenario, some authors have investigated utility-based forwarding [32][55][56] where the general approach is using a utility function to assign a certain data rate, depending on how useful the packets are in the algorithm. However, in a VANET, utility is assigned not to a data rate but to a data packet.

As such, when implementing congestion control through the use of utility functions, the node must calculate a certain utility value for each packet when a transmission opportunity arises.

Utility-based Routing Control

Once a node receives an advertisement packet and queries the membership of its own packets to the neighbour's bloom filter, this node is authorized to send packets. Using a utility function, the packets it contains in its storage are assigned, each, a utility value based on a function and then sorted from highest utility value to lowest. That utility value, $u_{TX}(p) \in [0,1]$, corresponds to the utility of transmitting this data packet at this point in time.

According to [56], a possible, simple utility function would be as shown in Equation 4.5, where a full file in a dissemination experiment has a certain number of file packets or segments and utility would, therefore, represent the percentage of the transmitted file.

$$u_{\mathrm{TX}}(p) = \frac{\text{"number of file segments"}}{\text{"total file segments"}} \tag{4.5}$$

In mOVERS, in order to facilitate the packet forwarding decision, the nodes contain an internal structure listing the packets and information associated to them, one of which is the utility value.

Upon bloom filter comparison, if a neighbour does not contain all files of the sender node, that is, if the sender node has any packet that the neighbour does not have, the sender node is able to send data packets. Once data packets are clear to be sent, the utility is calculated for each packet, based on the number of packets that the sender node has, and the packets are sent accordingly. After the contact, the storage is then re-sorted, considering the number of transmissions as the tie-breaker.

The end result before data transfer is a node clear to send packets to its neighbour with the guarantee that it will peek the packets of the highest utility to the neighbour during the transmission period.

4.4 mOVERS Integration

In order to implement the proposed content distribution strategy in the mOVERS software, several steps needed to be taken. For purposes of strategy testing and analysis, this is the crucial stage of this Dissertation. The following sub-section will describe the global modifications required to implement the proposed solution, as well as a description of the changes

implemented in the Routing module (and consequently, HandlerFILTER). Afterwards, a description of the implemented code and the consequential dissemination process will be provided through the use of flowcharts.

It is worth noting that the mOVERS' code as an emulator is equal to the mOVERS' code that is in the real OBUs and RSUs of the vehicular network. In this manner, it only takes compiling the code inside the boards for it to be tested in a real network.

4.4.1 Global Modifications

As illustrated in Figure 4.4, the Handler module is considered as a separate module from Routing. However, this is merely for purposes of illustration of the changes made to mOVERS. Ultimately, the Handler module is part of the Routing module. An auxiliary sub-module was added in the Handler module named **FILTER**. The Handler module contains all content distribution schemes and is now divided in 4 different ones: LNHF, LRBF and LRGF, which have been previously mentioned and FILTER, which contains the proposed content distribution scheme in this Dissertation. Both the Routing module and the mOVERS support library also suffered some changes, in order to handle the packets received from the HandlerFILTER functions.

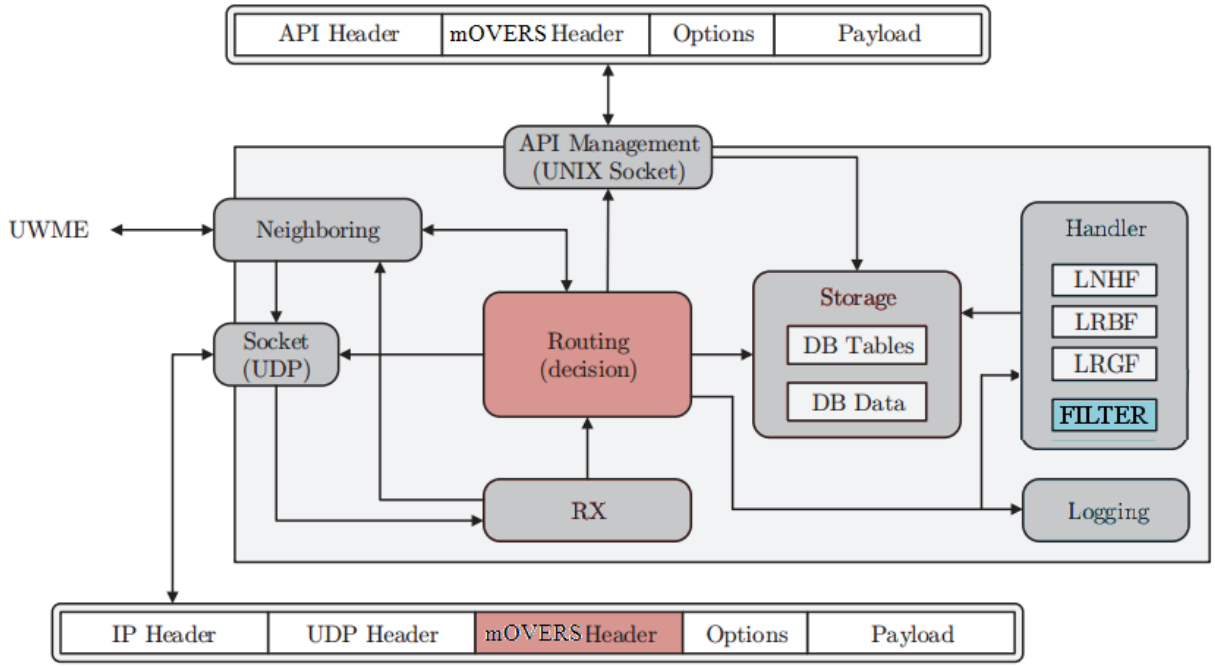


Figure 4.4: mOVERS Architecture (additions in blue, modifications in red), based on [16].

In the following sections, the changes made to the mOVERS Emulator to support the proposed content distribution scheme will be explained in further detail.

mOVERS Support Library

Along with the mOVERS software, the emulator has a support library called *libmOVERS*, which defines a few specifications and has a few functions to be used by applications. The key functionality of this library is the structure of the mOVERS packet header.

In order to integrate the proposed content distribution strategy, this header was modified by adding the following fields:

- *tableSize*: The size of the bloom filter;
- *numberOfElements*: The number of elements in a neighbour's Bloom Filter;
- *hashFunctions*: The number of hash functions used for the creation of a neighbour's Bloom Filter;
- *seed*: Value that ensures a constant behaviour of the hash functions;
- *desiredFPP*: The intended False Positive Probability of the neighbour's Bloom Filter.

4.4.2 Bloom Filter Strategy

In order to implement the proposed solution, a new class of objects was created, named HandlerFILTER. The Routing module creates an object of this class to perform content dissemination decisions. Alongside it, a previously implemented inner class called Node is also created. Each object **Node** contains a packet associated to it along with a hash identifier. For the purpose of this strategy, the class Node uses the following members: hash, fileID, nTx and utility. All content distribution packets have associated to them a hash, a correspondence to a file, a number of transmissions and a utility. In this class, the most relevant methods are the **bloomFilterTransmission** and the **bloomFilterReception**, responsible for instructing the machines on how to handle the transmission and reception of advertisement packets involving Bloom Filters.

The following sub-sections will further explain how the dissemination process is deployed in mOVERS.

Bloom Filter Structure

The implemented bloom filter is a class containing a structure with the following members: bitTable, numberOfHashFunctions, tableSize, numberOfElements, seed and desiredFPP. All of these members are initialized as zero and are filled when a bloom filter is created. The main methods involved in creating a bloom filter, filling these members are **computeOptimalParameters**, **effectiveFPP**, **insert**, **contains**, **hashing** and **saltGenerator**.

Whenever a new filter is to be created, some parameters need to be set *a priori*. Those are the number of elements, the false positive probability and the seed. The number of elements to be inserted is predetermined as the number of packets a node has in storage. The false positive probability needs only to be initialized, since its actual value will be computed after

the elements' insertion in the filter through the *effectiveFPP* method. Finally, an initial seed is selected, also for initialization. Once the number of hashing functions required for the filter is calculated, a corresponding number of random seeds is gathered to be used.

The *computeOptimalParameters* method will find the optimal minimum number of hash functions and minimum size of the bit table required, taking into account the number of elements to be inserted, through the use of Equation 4.1. The following algorithm will provide a brief explanation on the calculations performed (the number of iterations was deemed appropriate after experimentation).

Algorithm 1 Optimal Parameters

```
1:  procedure BEGIN
2:      while iterator < 1000 do
3:          numberHashes = iterator
4:          Calculate minSize
5:          if size < minSize then
6:              Assign current minimum as minimum size of table
7:              Assign current number of hashes as minimum number of hashes
8:          iterator++
9:      if size < minSize then
10:         size = minSize
11:     else if size > maxSize then
12:         size = maxSize
13:     if nHashes < minHashes then
14:         nHashes = minHashes
15:     else if nHashes > maxHashes then
16:         nHashes = maxHashes
```

The end result of the algorithm is the number of hash functions and table size to be used in the bloom filter. It is worth noting that the initialized size is the maximum value permissible by the program, so that it can constantly decrease until a minimum is discovered.

Both the *hashing* and *saltGenerator* methods are associated with the operations required to turn a packet's identifier into a bit to be inserted in a certain place in the bit table. The salt generator is responsible for generating different random seeds from a predetermined set so that a single hash function (present in the *hashing* method) can generate unique bloom filter instances when inserting an element into the array. The hash function used was the AP Hash Function suggested by Arash Partow in [57], which applies a hybrid rotative and additive hashing algorithm. The following equations are an algebraic description of the hash function:

$$h1_{\mathrm{i}}(m_{\mathrm{i}}) = h1_{\mathrm{i\text{-}1}}(m_{\mathrm{i\text{-}1}}) \oplus (h1_{\mathrm{i\text{-}1}}(m_{\mathrm{i\text{-}1}}) \ll 7) \oplus m_{\mathrm{i}} \oplus (h1_{\mathrm{i\text{-}1}}(m_{\mathrm{i\text{-}1}}) \gg 3 \tag{4.6}$$

$$h2_{\mathrm{i}}(m_{\mathrm{i}}) = h2_{\mathrm{i\text{-}1}}(m_{\mathrm{i\text{-}1}}) \oplus K_{\max} \oplus (h2_{\mathrm{i\text{-}1}}(m_{\mathrm{i\text{-}1}}) \ll 11) \oplus m_{\mathrm{i}} \oplus (h2_{\mathrm{i\text{-}1}}(m_{\mathrm{i\text{-}1}}) \gg 5) \tag{4.7}$$

$$H(m) = \sum_{i=0}^{n-1} h_{\mathrm{i}}(m_{\mathrm{i}}) = \begin{cases} h1(m_{\mathrm{i}}), \text{ if n is even,} \\ h2(m_{\mathrm{i}}), \text{ if n is odd,} \end{cases} \tag{4.8}$$

Essentially, each packet hash (8 bits) to be inserted in the filter will have an even number of specific bits, determined through left and right shifts a number of times, combined, until a single bit remains. As a result, this hash function decreases the size of the packet identifier further and further until only a single bit remains to be inserted in the filter. The number of resulting bits to be inserted in the filter is dependent on the number of salts, that is, the number of times a packet identifier must go through this process.

Finally, both the *insert* and *contains* methods work similarly by sending a packet's identifier through the hashing function a number of times (indicated by the salt generator) and filling the respective bits in the bit table. In the *contains* method, if a single bit inserted is different from the one already inserted there, the method returns false and, since the presence of the packet is inexistent in the filter (within reason of a false positive probability), it returns false; otherwise, it returns true. The following algorithms will provide a brief explanation on the insertion and querying of a packet in the bit table.

Algorithm 2 Insertion Algorithm

```
1: procedure BEGIN
2:     for salt=firstSalt to salt=lastSalt do
3:         Reinterpret packetHash through hash function
4:         bitIndex = packetHash % tableSize
5:         bit = bitIndex % 8
6:         Assign bit to the bit table
7:     Increment number of elements in the filter
```

Algorithm 3 Query Algorithm

```
1: procedure BEGIN
2:     for salt=firstSalt to salt=lastSalt do
3:         Reinterpret packetHash through hash function
4:         bitIndex = packetHash % tableSize
5:         bit = bitIndex % 8
6:         Assign bit to the bit table
7:         if Inserted bit != Current bit then return false
       return true
```

Beginning and End of Dissemination

Whenever a node wants to send advertisement and/or data packets, there are flags that must be set to *true* to allow so. For broadcasting advertisement packets, the flag **is2sendAdv** is, in the beginning of dissemination, set to false in OBUs and set to true in RSUs. When this flag is enabled, a node is capable of sending advertisement packets to its neighbours. For broadcasting data packets, the flag **returnPacketFlag** is set as false in both RSUs and OBUs and, when enabled, allows nodes to exchange data packets.

As a result, when mOVERS starts running an experiment, only RSUs are allowed to send advertisement packets to OBUs and no machine is able to transmit data packets. RSUs are periodically sharing the Bloom Filter created by them. When an OBU that has no content is within range of an RSU, it receives this advertisement packet and knows that the RSU has content to be received. Therefore, it updates its flags and is able to broadcast advertisement packets.

The RSU, in return, receives an empty Bloom Filter from the OBU and, by extension, knows that the neighbour has no content that the RSU contains. Therefore, the RSU changes its flags' value to enable the broadcast of data packets.

The end of a node's dissemination process happens each time a node has no neighbours or if the neighbours' content proves to be equal to the sender's node. In such a situation, the flag enabling the broadcast of data packets is set to false. Figures 4.5 and 4.6 illustrate this procedure.

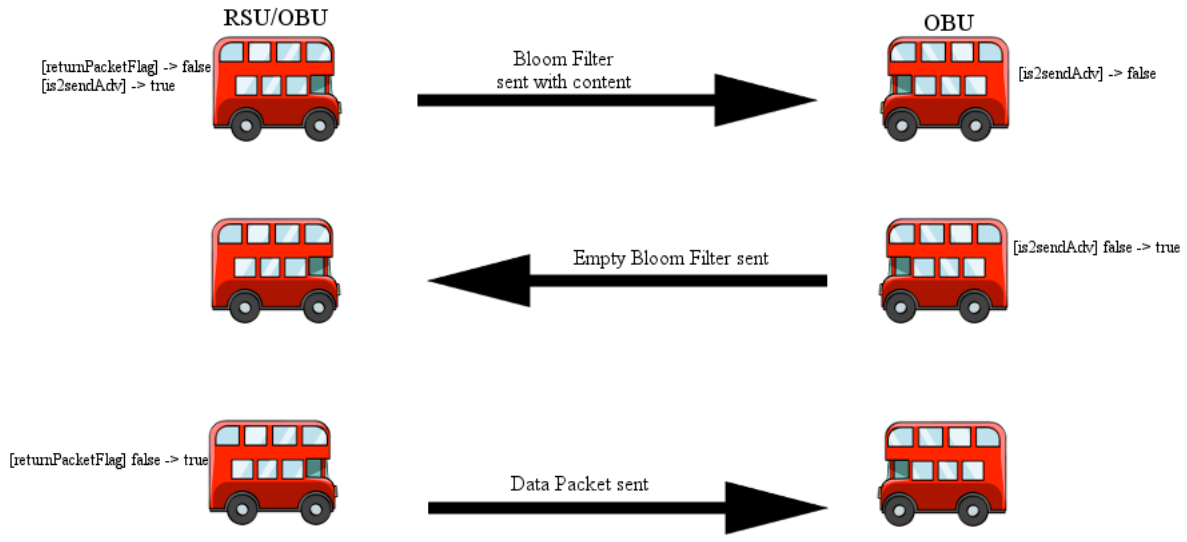


Figure 4.5: mOVERS Start of Dissemination

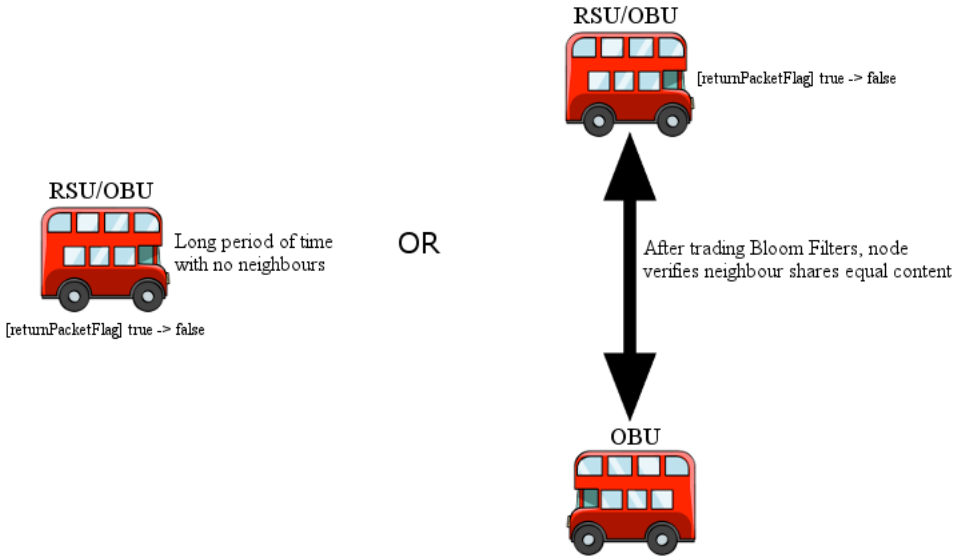


Figure 4.6: mOVERS End of Dissemination

Advertisement Packet Forwarding

This content distribution strategy relies on using Bloom Filters as a form of summarizing a node's storage contents in order to decrease the size of advertisement packets and, therefore, the network's overhead. As such, the information in this packet will be fundamentally different from previous strategies involving the use of a control packet. The only data that will be introduced in the advertisement packet as payload will be the bloom filter's bit array, already with the information of the sender node inserted into it. However, in order to use functions of comparison and hashing, the receiver must be able to replicate the whole filter and its properties, not just the bit array. Therefore, the header of the advertisement packet will contain necessary information for such effect. The structure of the advertisement packet is as observed in Figure 4.7.

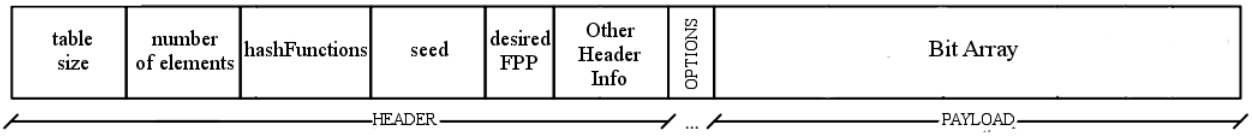


Figure 4.7: HandlerFILTER Advertisement Packet Structure

In order to understand the procedures for advertisement packet forwarding, a flowchart is presented in Figure 4.8. If the flag is2sendAdv is set to true, advertisement packets are to be sent. As such, an advertisement packet containing the node's bloom filter needs to be built,

using the bloomFilterTransmission method. This method starts by obtaining the number of projected elements that are to be inserted into the Bloom Filter. That number is always the number of packets that a node currently has in its storage. The false positive probability desired is also requested as an initial value for optimal parameter calculation. That is only an initial value. The actual false positive probability is computed using Equation 4.1.

Once these parameters are set, the node calculates the optimal table size and number of hash functions that the filter must have, in order to fulfill the first set of parameters. Once these parameters are created, the node will iterate its storage for each of its packet's identifier and insert it into the filter. Filter insertion happens by grabbing the packet identifier and hash it a number of times equal to the number of hash functions chosen as optimal parameter. Each bit resulting from each hashing will then fill a bit of the created array. Once the filter is complete with all the inserted elements as bits in the array, the header is filled with the relevant information to recreate the filter and the payload is filled with the bit table. Afterwards, the advertisement packet is sent by broadcast and Logging methods record the number of sent advertisement packets as well as the size of each packet.

The sending of advertisement packets is controlled by a thread that is invoked in a random periodical way between a minimum and a maximum value, MIN_CONT_ADV_PERIOD and MAX_CONT_ADV_PERIOD, respectively, until a cleaning signal is sent.

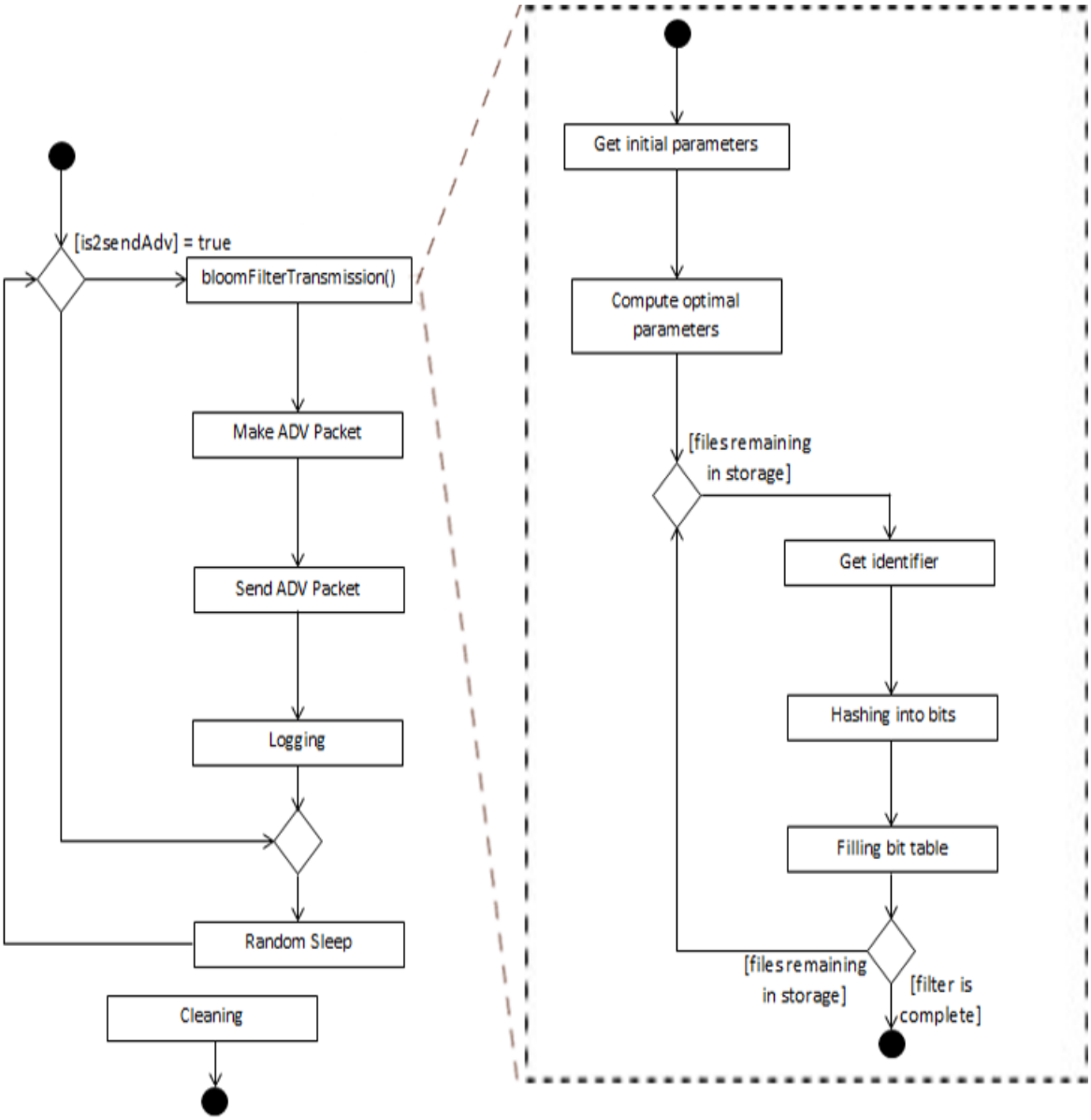


Figure 4.8: Advertisement Packet Forwarding Flowchart

Advertisement Packet Reception

In order to understand the procedures for advertisement packet reception, a flowchart is presented in Figure 4.9. Upon reception of a bloom filter-based advertisement packet, the receiver node must first confirm if the neighbour contains relevant information to it. For that, it must replicate the neighbour's filter. Firstly, the receiver node extracts relevant information to the filter from the packet header, namely the number of elements, the false positive probability and the seed of the neighbour's filter. As a result, the receiver can now replicate the sender's bloom filter. However, this filter is created with an empty bit table since nothing was inserted. Secondly, the bit table is then extracted from the payload and acts as part of the created filter. The end result is a replica of the sender's bloom filter in the receiver node. The following algorithm will provide a brief explanation on replicating a neighbour's bloom filter.

Algorithm 4 Bloom Filter Replica Algorithm

```
1: procedure BEGIN
2:     Received advertisement packet
3:     Retrieve neighbour's bit table from Payload
4:     Obtain neighbour's initial parameters and used seeds from header
5:     Build empty filter with neighbour's parameters
6:     Replace calculated seeds with neighbour's seeds
7:     Copy neighbour's bit table to empty filter
```

Finally, the receiver node must insert its own elements into the filter and verify if there is at least one of the neighbour's elements that does not match the receiver's packets. This comparison operation is performed in bloomFilterReception using the method *contains* mentioned in Algorithm 3. The receiver node iterates its storage for each of its packet's identifier, hashes it a number of times equal to the number of Hash Functions used (also extracted from the packet's header) and inserts it into the filter, using the method. This method will return false if even a single bit that is to be set at '1' is set at '0'. Such will mean that the receiver node contains at least one packet that the sender does not have (within likelihood of a false positive).

Given this situation, the receiver updates its flags to enable broadcast of advertisement packets as well as data packets, since it has both received its first advertisement packet and has established that there is need for its content.

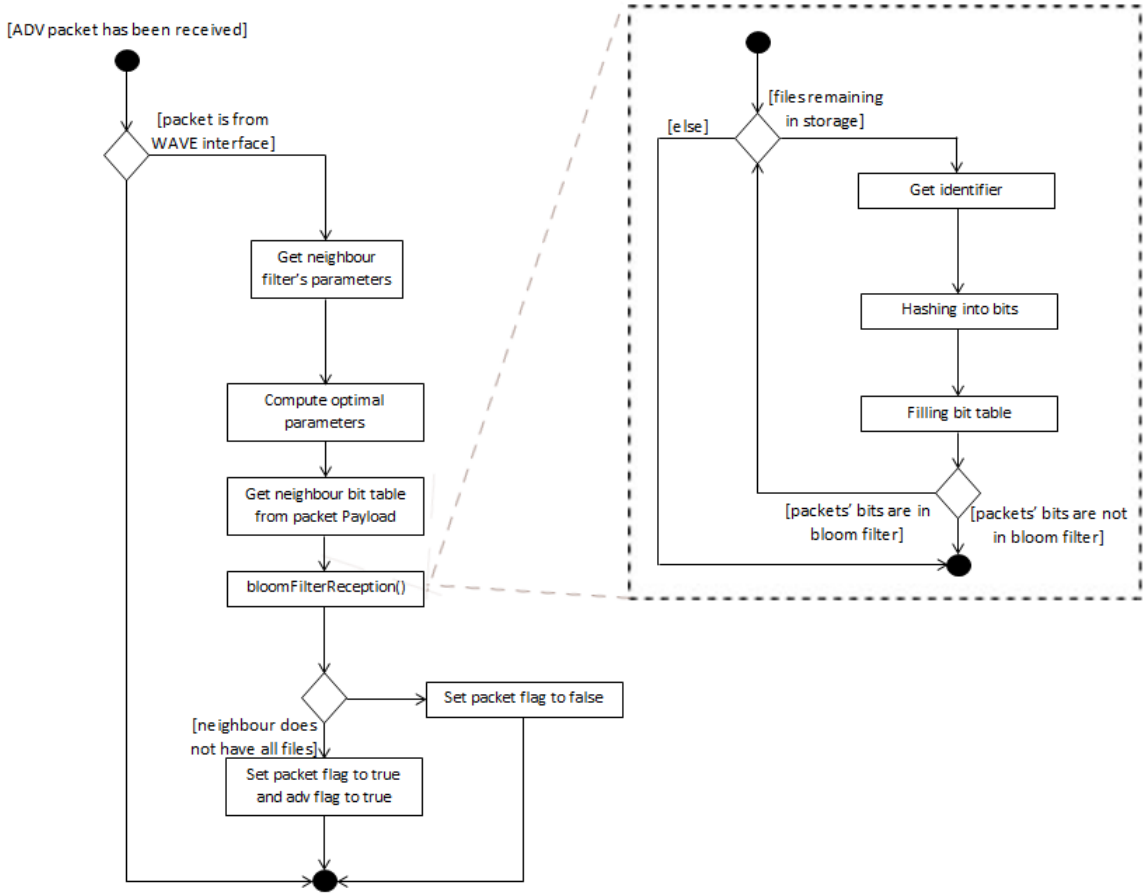


Figure 4.9: Advertisement Packet Reception Flowchart

Utility Field

As previously mentioned in Section 4.3.3.2, a utility function can be as simple as Equation 4.5, establishing that the more complete a file is, the more relevant this node's contents are to its neighbours. A new member was added to the class Node named **utility**. In this manner, each data packet always has a *utility* value attached to it.

Furthermore, some simple methods were added to handle and update the utility values of the packets in a node's storage. Those methods are **addUtility**, **updateUtility** and **setNodeUtility**. *addUtility* is the main method handling utility of packets. It is invoked each time a new packet is stored in a node and each time an advertisement packet is received. If a new packet is associated to a new file, it calculates its utility value and invokes *setNodeUtility* to associate that value to the packet's data structure. If it is from an already known file, then all packets corresponding to such file are updated. Therefore, *updateUtility* is invoked. Upon advertisement packet reception, all packets' utility is re-calculated for potential data packet broadcasting.

Data Packet Forwarding

The flowchart in Figure 4.10 illustrates how data packet forwarding operates in mOVERS. The strategy used here is analogous to LRBF's data packet forwarding. If the sender node has only OBUs as neighbours and contains packets in storage, it will peek the first hash, representing the first packet to be loaded from storage. If no hash is found, the associated Node must be deleted. After the hash is retrieved, the corresponding packet is sent in broadcast and the number of transmissions of such packet is incremented. Then, the storage is sorted since the number of transmissions is considered a tie-breaker if utility is equal. At the end, Logging methods will keep information on transmitted packets.

Data Packet Reception

The flowchart in Figure 4.11 illustrates how data packet reception operates in mOVERS. Much like data packet transmission, the strategy is very similar to LRBF. Since dissemination is a downloading process, only OBUs are considered as receiver nodes. If the packet was received from a WAVE interface, the packet's hop list is updated with the ID of the receiver. Upon packet reception, if there was no prior knowledge of that packet, a Node is associated to it, regarding packet utility, number of transmissions, among other information. However, if the associated node already existed in the receiver, the corresponding Node's number of neighbours is updated. Afterwards, the packet is stored (only if it was not already) and the Logging records information regarding stored packets.

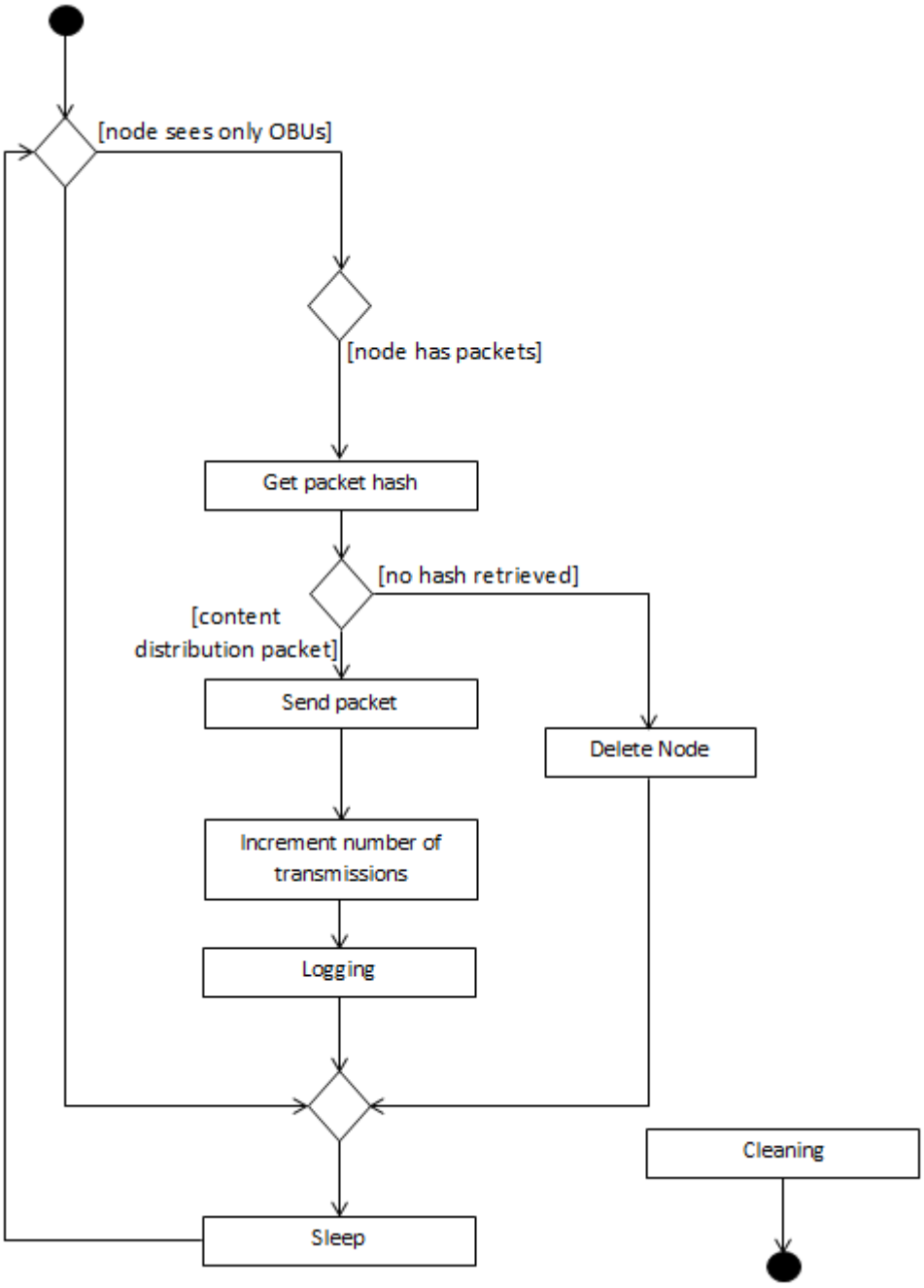


Figure 4.10: Data Packet Forwarding Flowchart

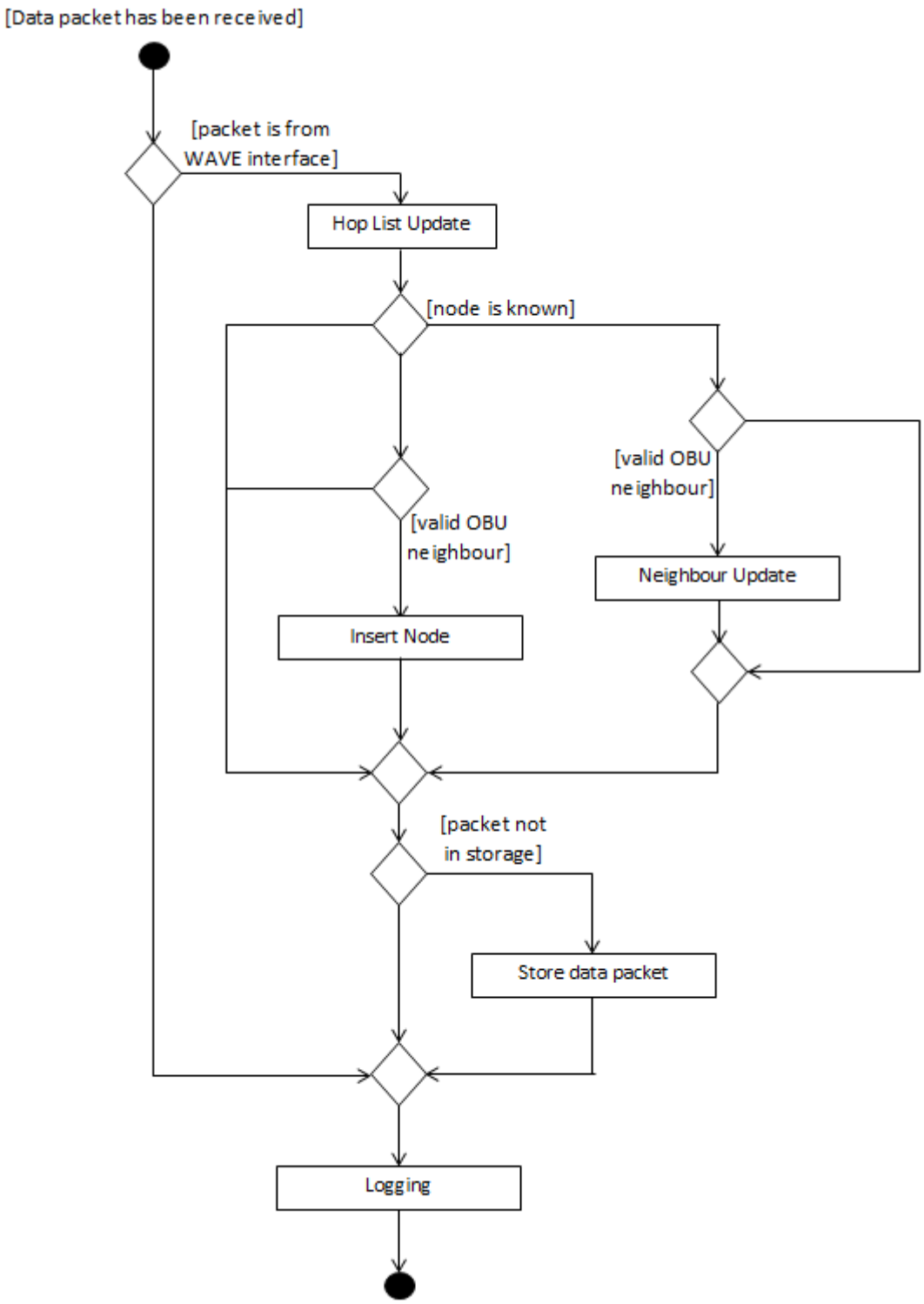


Figure 4.11: Data Packet Reception Flowchart

4.5 Summary

This chapter focused on describing the implementation of the proposed content distribution strategy. From the theoretical notions on Bloom Filters to Utility-based congestion control, all the main points of this Dissertation were covered. After such, integration of said strategy was described, from the global modifications required in mOVERS to the bloom filter and utility algorithm. The following chapter will provide the reader with the results obtained from mOVERS and discussions based on statistical analysis.

Chapter 5

Tests and Results

5.1 Chapter Description

Once designed and implemented, the content distribution strategy needs to be evaluated in the development platform mOVERS already described in Chapter 3. As such, this chapter presents the equipment used in the evaluation, the evaluated scenarios and the main results obtained in them.

This chapter is organized as follows.

Section 5.2 will describe the specifications of the equipment used for the development of this Dissertation.

Section 5.3 will provide a global spectrum of the vehicular network, outlining the scenario in which real data was obtained for emulation purposes.

Section 5.4 will present the Bloom Filter's performance when used in the proposed content distribution strategy.

Section 5.5 will present the main results achieved after a set of experiments performed in the mOVERS software.

Section 5.6 summarizes this full chapter.

5.2 Equipment

The content distribution strategy was developed and modified in a personal laptop. However, the emulator was then uploaded to a different computer that ultimately ran the content dissemination experiments. The specifications of the machine that ran mOVERS are as described in Table 5.1.

Operating System	64-bit Ubuntu 14.04 LTS
Processor	Intel® Xeon® CPU E5620 @ 2.40 GHz x8
RAM	12 GB

Table 5.1: Machine used to run mOVERS.

Due to the heavy computational power required to run mOVERS along with the long emulation time, a Virtual Machine was used within the server mentioned in 5.1. This Virtual Machine was configured using VMware Workstation 12 Player and was accessed via command line to run the tests. It is noteworthy that this server would share its resources with other processes that may occur. As such, all performance metrics regarding the machine must not be seen as absolute, since these will vary, each time the same experiment is carried.

5.3 Vehicular Network Scenario

The city of Oporto is home to a vehicular network, interconnected to Porto Digital's fiber network [58] that has been central in projects with the University of Aveiro, University of Porto, IT and Veniam®. In order to better understand the network topology and nodes' behaviour, two 24-hour data collections have been provided, one of October 31st, 2014 and one of February 13th, 2015. For statistical analysis purposes, only the most recent data was selected. Figure 5.1 shows the area covered by the vehicular network's infrastructure.

Figure 5.1: Oporto's vehicular Network, created using [59].

The 24-hour collection used in the mOVERS tests contain information on the location of all the RSUs and more along the coast of Portugal. However, since the emulator is quite heavy in resource consumption on the machine that is running it, a smaller number of RSUs was used in the simulations. An analysis of the RSUs' location shows that the optimal location to test content distribution algorithms while, at the same time, limiting their number, is an

area centered around São Bento and Aliados, as shown in Figure 5.1 with a circle. Despite the limitation of the number of machines, strategy evaluation is not compromised since the nodes' behaviour is most focused around the chosen area and contacts are far more sparse away from that area.

Based on a study of the Oporto network's vehicle routes by [16], two timeframes were selected to evaluate the proposed routing algorithm. Those are the 6h-10h timeframe, which is denominated as the **Rush Hour** and the 10h-14h timeframe, designated as **Non-Rush Hour**. Furthermore, though the vehicular network contains data on over 300 OBUs and 50 RSUs, the emulator is incapable of emulating such a high number of nodes or an excessively high congested network, leading to a potential Virtual Machine shutdown. As a result, both tests were run with 18 RSUs and Rush Hour and Non-Rush Hour's OBUs numbered 161 and 133, respectively. Moreover, messages are only sent whenever the RSSI signal is, at least, 15 dBm.

5.4 Bloom Filter Performance

To ensure that the bloom filter has a proper behaviour, its false positive probability was logged during a 4-hour experiment between 6:00 AM and 10:00 AM, the timeframe of most vehicle density due to rush hour. Figure 5.2 illustrates the evolution of the false positive probability of the bloom filters created throughout the experiment.

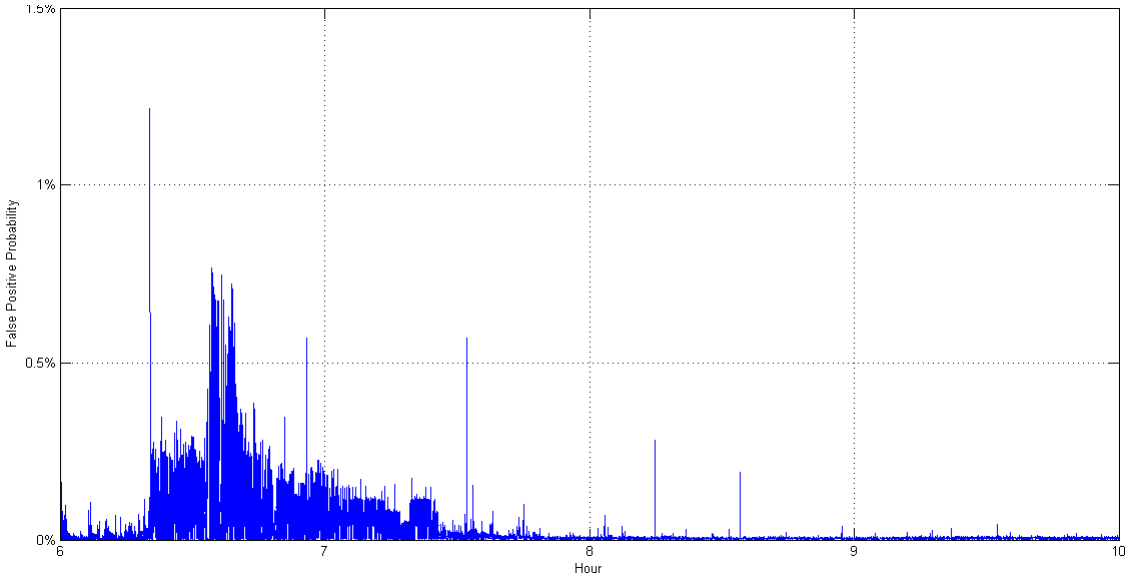


Figure 5.2: Bloom Filter's False Positive Probability

As observed, throughout all timestamps of the experiment, the false positive probability rarely exceeds 1%, making it a reliable data structure to act as a routing decision maker for this content distribution strategy. The following section will provide results of the tests made with this strategy in the mOVERS software.

In order to ensure that there was no decisive factor that greatly changed the filter's performance, a script named *bloom_filter_example* was used to test proper behaviour of both the functions of insertion and containment as well as its response, depending on the seeds used. Given that the filter uses a single hash function with different seeds, the effect of the seed parameter in the filter is merely to add "garbage" to the function, so that a single hash function can behave like multiple hash functions, hashing each bit in a different place in the filter. Regardless of the seed used, the filters obtained always show the same false positive probability and proper insertion and containment functionalities.

5.5 mOVERS Emulator Tests

A content distribution strategy based on a Bloom Filter has been implemented and evaluated via mOVERS. For comparison purposes, our Filter strategy will be evaluated next to the strategy that had the best delivery ratio up to date, LRBF (Local Rarest Bundle First) as well as LRGF (Local Rarest Generation First), a first attempt at limiting network overhead.

All experiments considered the dissemination of a 75 MB file divided in 2256 packets of 32 KB each, and were run 3 times. The results, whenever possible, represent the mean and 95% confidence levels. It is worth pointing out that the mOVERS emulator has a very long running time. Each content dissemination strategy test takes 20 hours to complete, making mOVERS impractical for a large number of content distribution emulation tests. This heavy resource consumption is due to the fact that each machine (OBU and RSU) is run as a separate process within the server running mOVERS. As such, a single server has to run nearly 200 machines at the same time. As a result, running more tests to obtain more robust confidence intervals was not possible.

5.5.1 Rush Hour

This is the timeframe between 6:00 AM and 10:00 AM, the period of highest vehicle density. As such, this is the most important timeframe to perform tests, due to the vehicles' high mobility, in establishing an ideal DTN scenario to test our content distribution strategy. The following subsections will showcase several metrics used to verify the robustness of the proposed solution.

File Distribution

File Distribution is associated with the evolution of metrics regarding the percentage of packet delivery to the network, that is, how much was downloaded in the network. Such is obtained by illustrating the number of unique packets stored per OBU per hour in percentage. Figure 5.3 illustrates the evolution of our strategy, when compared to LRBF and LRGF.

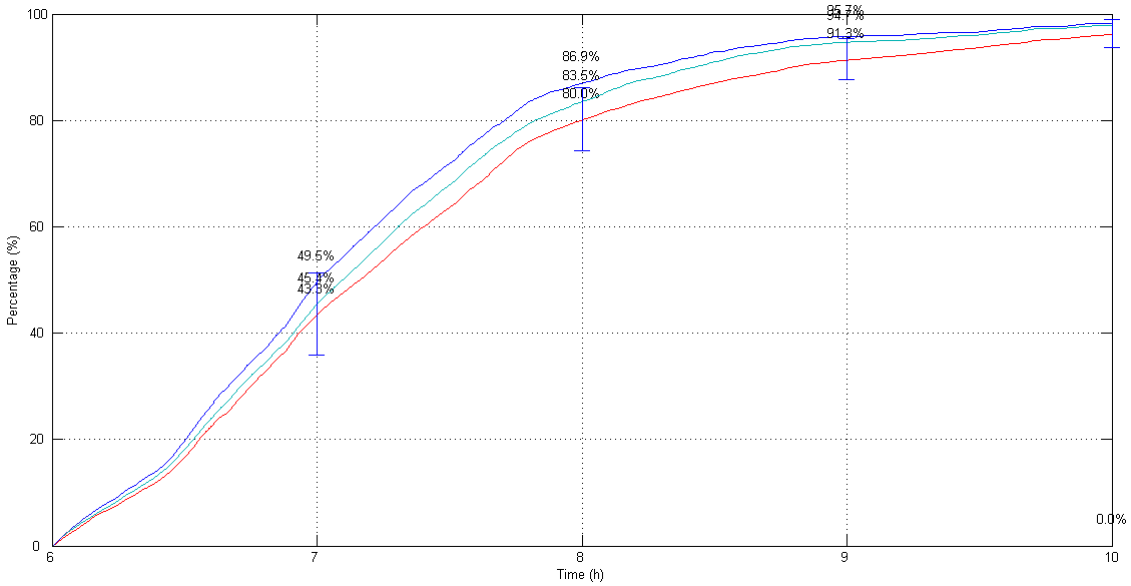


Figure 5.3: Percentage of File in Network

The results clearly show that both LRBF and LRGF are slightly superior in terms of delivery rate. LRBF has the highest delivery rate, primarily due to the fact that all nodes know the complete information of all of the neighbours' packets, being the upper bound for delivery ratio. LRGF limits that knowledge by having the nodes know information regarding only the rarest block of packets. When that information is even more summarized, in this case, being a "mathematical approximation", coupled with a basic utility function for storage sorting, FILTER's behaviour becomes slower. Not only there is less specifications in the advertisement packets sent, the nature of the utility function makes it so that, as certain packets become more useful, others end up remaining at the "bottom" of the storage and may never be sent, leading to the small loss in file distribution of 3-6%. However, such loss is a trade-off, since the algorithm becomes highly efficient in overhead size, as it will be proven in subsequent subsections.

After considering a 95% confidence interval, it can be seen that previous strategies fall within the potential interval of the proposed strategy during most of the dissemination process.

Progress Rate

Progress Rate is the metric that determines how fast the full content is being disseminated to the nodes. Figure 5.4 showcases this metric by showing the progress as the number of nodes that achieve the complete file every 10 minutes.

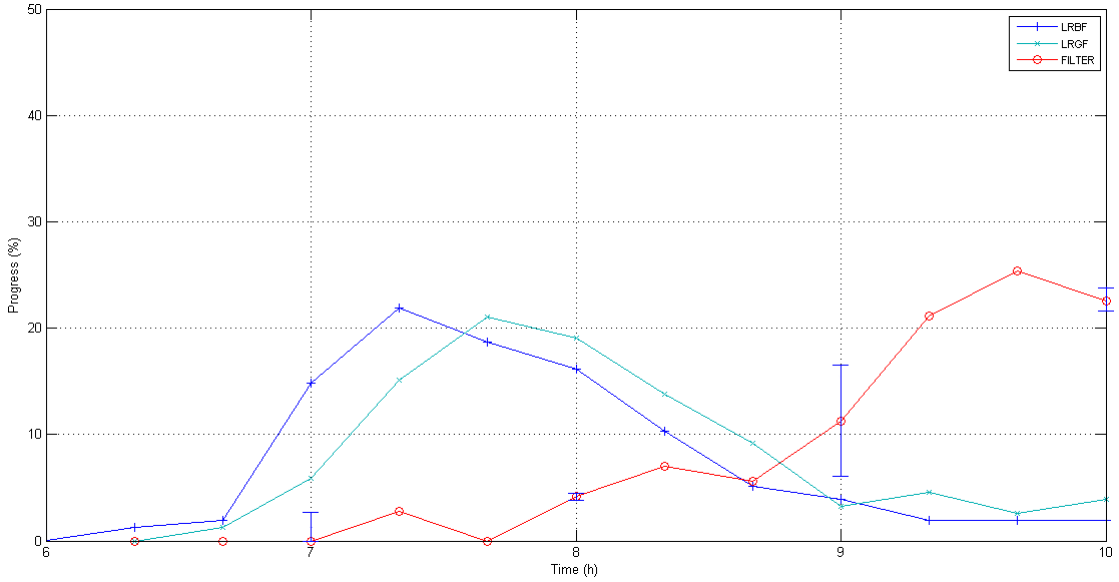


Figure 5.4: Progress Rate

The progress rate shows that LRBF and LRGF are faster in the second and third hours when downloading the complete file, yet its speed greatly decreases as the experiment continues. Such is due to the nature of the algorithm, greatly limiting transmission when most of the content has been downloaded. At that point in time, nodes are only looking for specific rarest packets in the network to complete the file.

On the other hand, FILTER is definitely slower in its download progress since most of its downloads are achieved in the third and fourth hours. This is attributed to the basic utility function used as the sorting function for the storage. Due to the nature of the utility function, all packets have roughly the same utility in the beginning and can, therefore, be disseminated equally. As such, there is no clear, optimized form of packet sorting to ensure delivery of only specific packets to ensure file completion at the first half of the experiment, and the nodes take a longer period of time to complete the file. Moreover, although it only surpassed 1% once, the false positive probability has shown that, roughly, during the first hour, the false positive probability is much greater than during the rest of the experiment, which can also be a factor, since it could cause repeated packets to be transmitted.

However, as vehicles slowly obtain packets and files start gaining priority, the algorithm quickly speeds up in the last two hours and achieves a good delivery ratio. However, because the progress rate is slower, despite all nodes having obtained roughly 95% of the file, a much lower amount of vehicles actually obtained the remaining 5%, completing the file. The fol-

lowing subsection will provide information on the number of vehicles that actually achieved the full 100% of content in the network.

Vehicles with Full Content

This subsection will provide the reader with insight on the number of vehicles that were actually able to obtain 100% of the information in the network. Such can be analyzed in Figure 5.5.

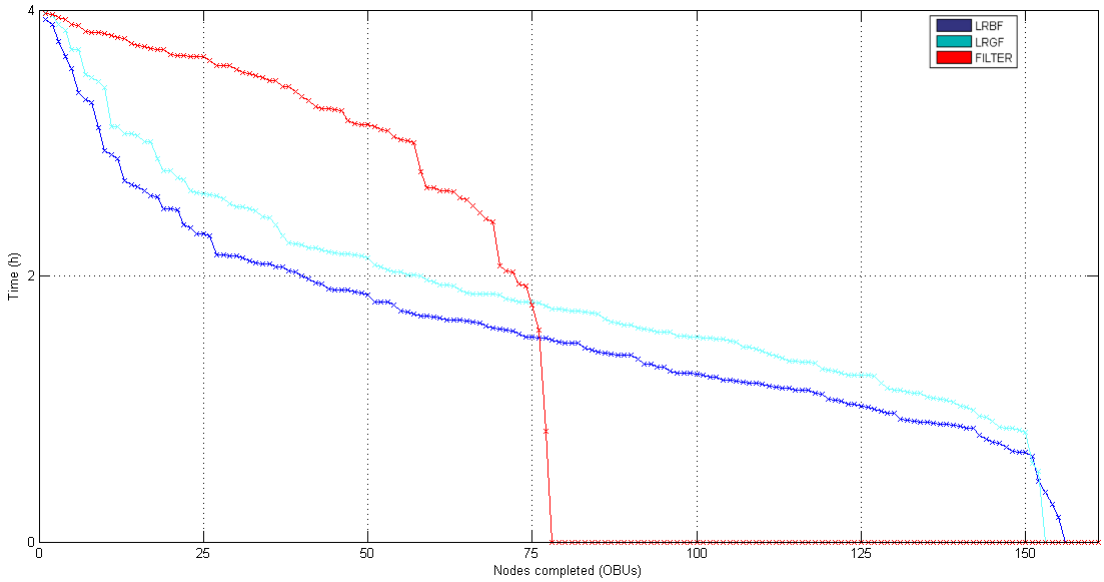


Figure 5.5: Number of OBUs with all content

Strategy	Average (h)	C.I. (95%)
LRBF	1.54	±0.14
LRGF	1.85	±0.14
FILTER	2.55	±0.18

Table 5.2: Statistics of Number of Vehicles with Full Content.

As derived from the Progress Rate, due to the lack of information regarding the packets when sending an advertisement packet, along with the fact that the utility function causes some packets to always remain at the "bottom" of the storage, a much lower amount of vehicles end up achieving 100% of the information.

Number of Advertisements

Advertisement periodicity is a random value chosen between 5 and 10 seconds, by default. This and the following subsections have the purpose of evaluating the impact of the

advertisement packets in the network. Mainly, how effective FILTER is in reducing advertisement packet overhead in the network. Figures 5.6 and 5.7 show the number of advertisement packets present in the network during the experiment.

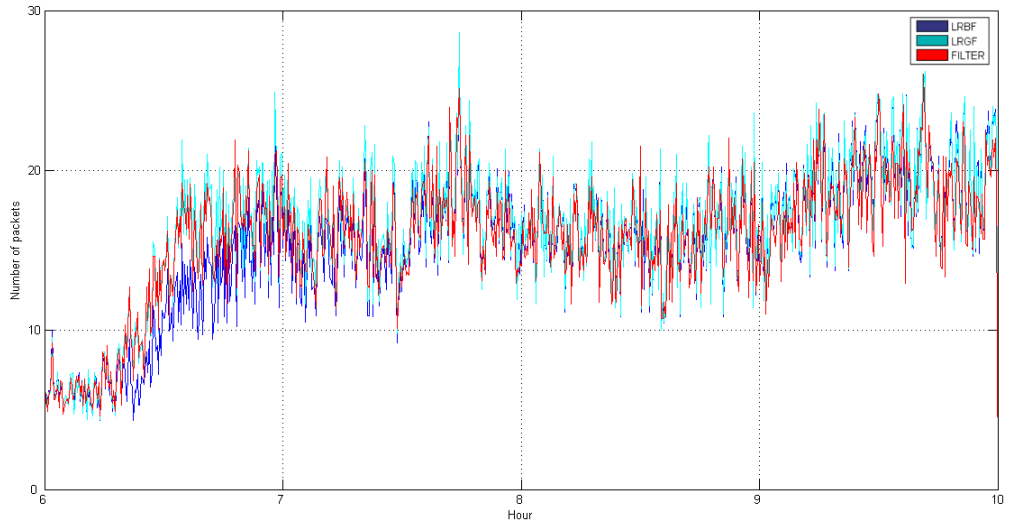


Figure 5.6: Number of transmitted advertisement packets (per timestamp)

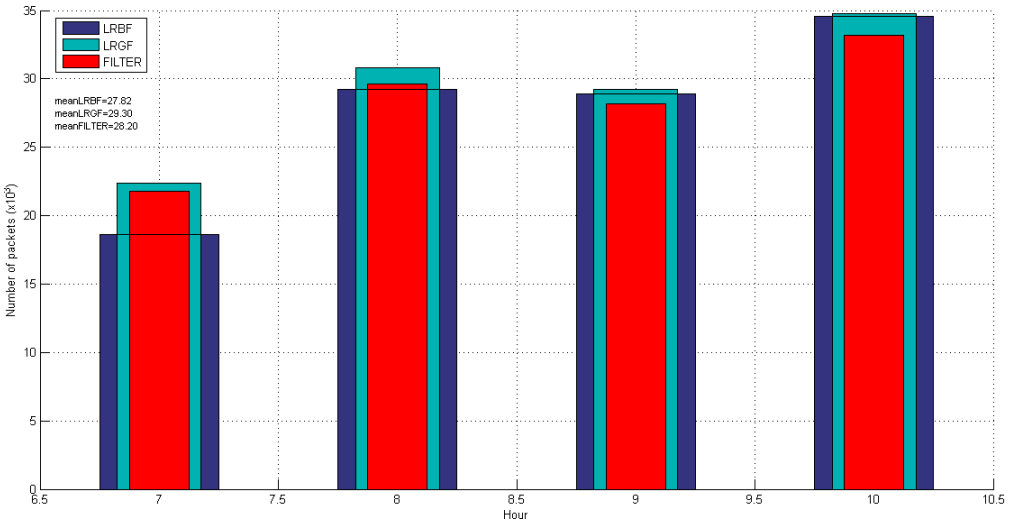


Figure 5.7: Number of transmitted advertisement packets (per hour)

Although a difference exists between the number of advertisement packets sent between the three strategies, the average number of advertisement packets sent rounds 28 thousand packets per hour. It was not expected that the strategy would have an influence on the number of advertisement packets themselves and this metric served to corroborate this fact.

Size of Advertisements

Figures 5.8 and 5.9 illustrate the size of the transmitted advertisement packets in the network during the experiment. Figure 5.8 shows that during the first two hours, LRBF shows a larger overhead than in the remaining two hours, which is to be expected since most of the dissemination process for LRBF occurs in the first half of the experiment. LRGF shows a similar behaviour, only with lower overhead than LRBF. FILTER, on the other hand, remains roughly constant in the size of its advertisements since, ultimately, it is only transmitting bits in the advertisement packet's payload, leading to a much lower network overhead.

Such statement is even more evident by observing Figure 5.9. During the first two hours, LRBF reaches nearly 60 MB of network overhead, which is too high for a vehicular network aiming to disseminate a 75 MB file. Even though it greatly decreases in the second half of the experiment, the average size of its advertisement packets reaches 31 MB. On the other hand, FILTER evidences that the size of the advertisement packets increases over time, which is as expected, since the progress rate showed that most of the dissemination process happens in the latter half of the experiment. However, even if the vehicles contain more packets over time, the size of the advertisements does not change as significantly as with LRBF, culminating with an average overhead of 13 MB, less than half of LRBF's overhead. LRGF, the initial solution to the overhead, stands as a middle ground, with an average overhead of 21 MB.

Furthermore, given that the exact same experiment was run multiple times, the variation in both the number and size of sent advertisement packets is minimal, as seen in Table 5.3.

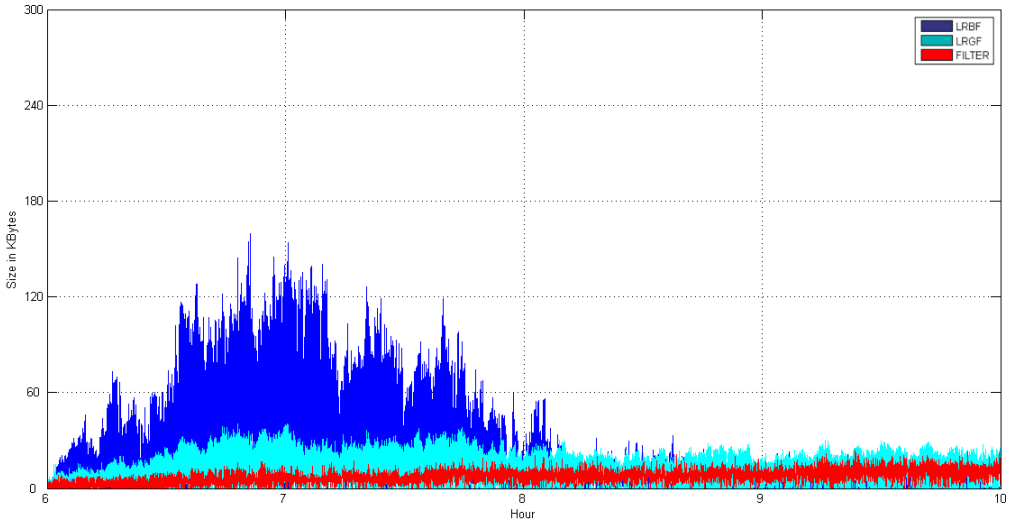


Figure 5.8: Size of transmitted advertisement packets (per timestamp)

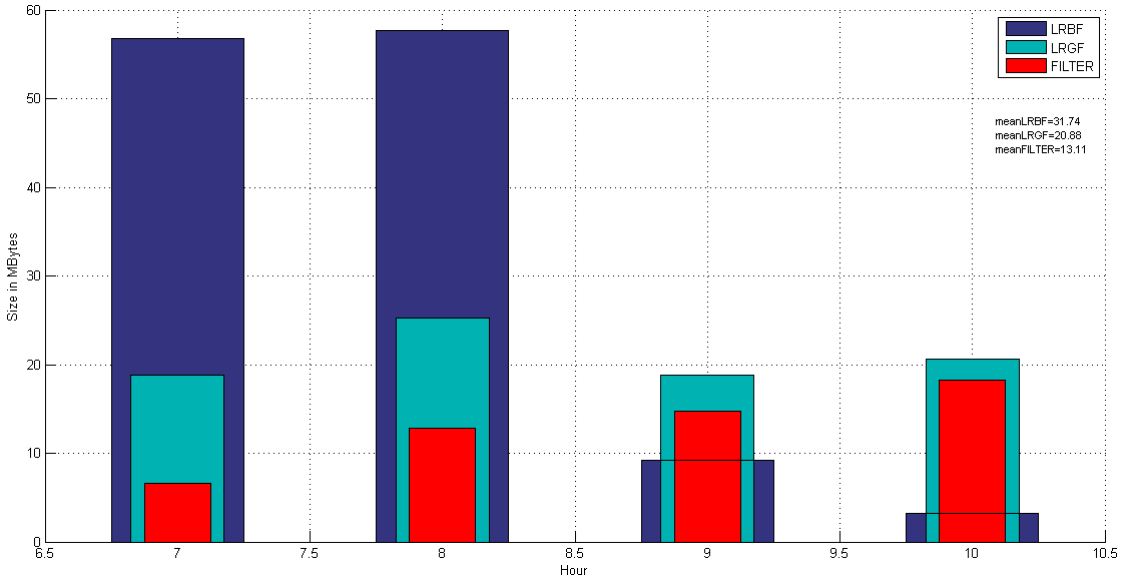


Figure 5.9: Size of transmitted advertisement packets (per hour)

FILTER Strategy	C.I. (95%)
Number	±0.03
Size	±0.01

Table 5.3: Rush Hour Advertisement Packet Statistics.

Server Performance Metrics

Finally, Table 5.4 resumes the physical performance of the server that runs the mOVERS emulator. Three metrics are presented and ascertain the machine's correct behaviour.

Strategy	CPU(%)	CPU std(%)	Load	Load std	RAM(%)	RAM std(%)
LRBF	10.1	8.2	1.2	1.0	46	7.6
LRGF	8.2	2.9	2.5	0.8	46.7	7.1
FILTER	27	4.9	5.9	1.9	35.8	9.8

Table 5.4: Computational Performance in Rush Hour.

A Bloom Filter is able to transmit the same information without consuming as much memory as the other two strategies, since all the information is coded into bits. However, it is very demanding in computational resources due to the heavy hashing of the packets required. As a result, it consumes far more processor effort and cores of the running machine, due to the amount of bloom filters and, consequently, the great amount of hashing calculations required per bloom filter per packet.

5.5.2 Non-Rush Hour

This is the timeframe between 10:00 AM and 14:00 PM, a period of lower vehicle density. Despite not as relevant a timeframe as the previous one, the following analysis aims to verify if the behaviour of the nodes in transmitting advertisement and data packets is, at least, similar. The following subsections will showcase the same metrics as before in the previous section.

File Distribution

Given that the metrics for evaluation are equal to the previous timeframe of study, this subsection will present once again an overview of how much information was effectively transmitted to the network's vehicles. Figure 5.10 illustrates the evolution of the FILTER strategy, when compared to LRBF and LRGF.

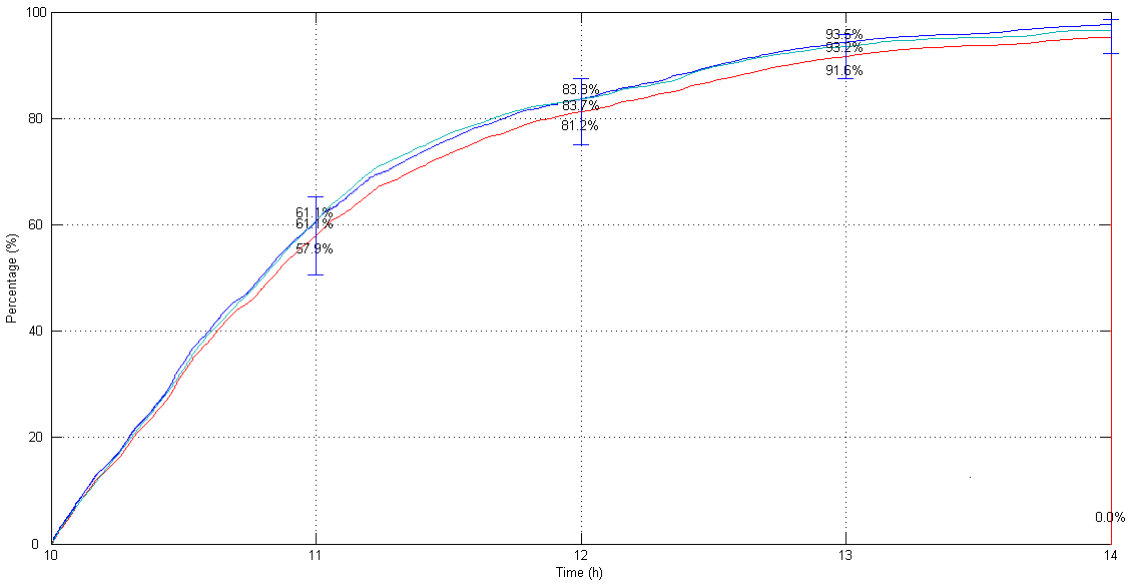


Figure 5.10: Percentage of File in Network

When compared to the Rush Hour timeframe, although LRBF still remains the superior content distribution strategy in terms of delivery rate, it is worth noting that both the LRGF and FILTER algorithms approach the same delivery rate more closely. Since the proposed dissemination algorithm remains the same, regardless of the studied timeframe or number of nodes in the network, the reason for the small discrepancy in delivery rate can, once again, be attributed to the storage sorting algorithm present as a Utility Function. Despite the network being more sparse, during the first hour, the network nodes are able to receive more packets than in the Rush Hour. This is due to the fact that during this timeframe, OBUs show a traveling route passing from parking lots to the city center, increasing the number of contacts with RSUs, increasing the delivery rate to the network.

Nevertheless, the network's delivery rate differs by such a small percentage (lower than

2%) that the proposed content distribution strategy is to be a preferred option, as it shall be shown in the subsequent subsections.

Much like the Rush Hour's experiment, the 95% confidence interval of the proposed strategy includes the previous strategies' performance, further strengthening the fact that this strategy approaches previous strategies closely.

Progress Rate

This is the metrics that illustrates how fast content is disseminated to the network's vehicles at a given time. Figure 5.11 showcases such metrics.

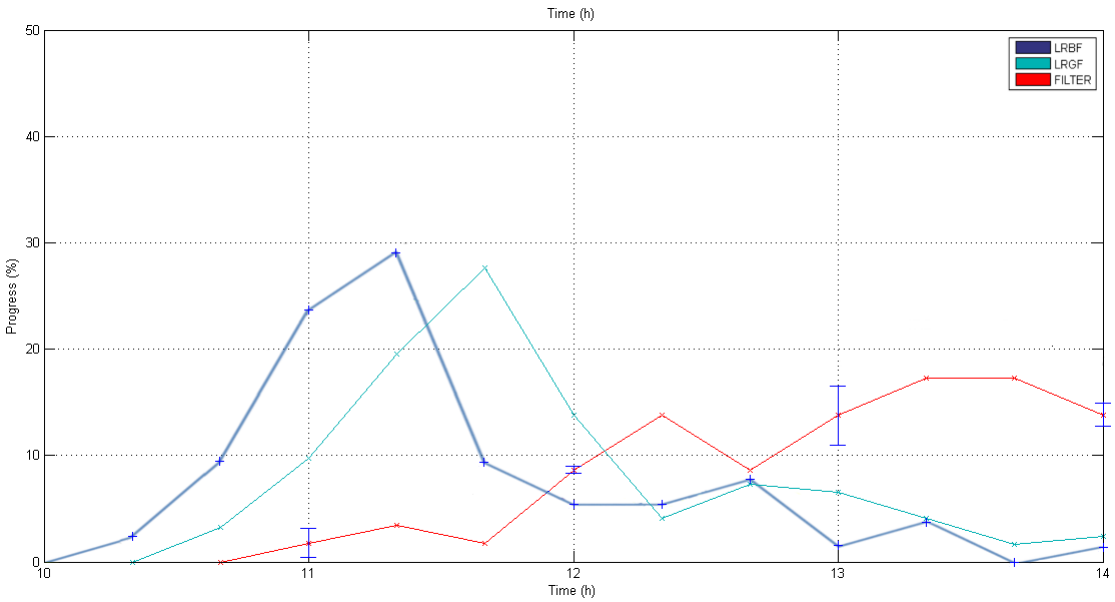


Figure 5.11: Progress Rate

It is again shown that LRBF is capable of disseminating most of its information in the first half of the experiment. This is, again, attributed to the intelligence present in the algorithm that reassures only specific packets are to be transferred. LRGF follows the same pattern albeit at a slightly slower rate. FILTER, on the other hand, once the first hour passes and some packets are transferred, its progress rate remains fairly constant, ranging only from 10% to 20%. Yet, the file is distributed in the network with nearly the same performance as LRBF. This can be attributed to the smaller number of vehicles present in the network, leading to a lower number of contacts and, by extension, a smaller network. Since the same number of RSUs was used to distribute information to a lower number of OBUs, the full network receives content faster.

The conclusion derived in the Rush Hour is not to be neglected, nonetheless. The first hour has very slow dissemination due to two possible factors, the utility function's simplicity and the false positive probability during that time. As a result, as seen in the following sub-

section, a much lower number of vehicles will be able to receive all the information in the network.

Vehicles with Full Content

This subsection will, once again, provide the reader with insight on the number of vehicles that were able to receive 100% of the information in the network. Due to the utility function's nature, some packets will always remain in the "bottom" of the storage and only the nodes that received those packets early on will be able to receive all content. Such can be analyzed in Figure 5.12.

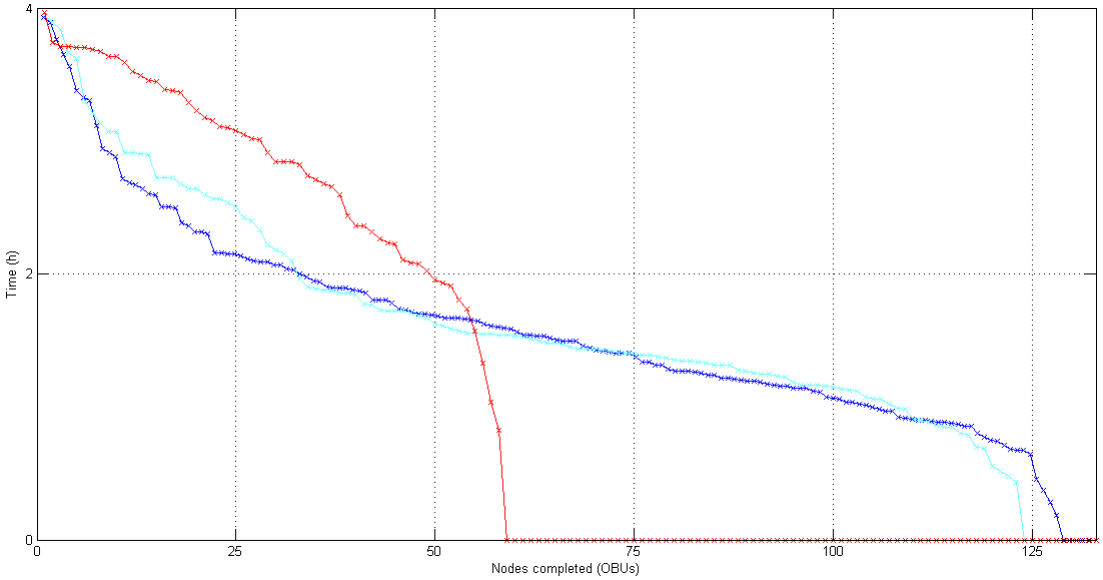


Figure 5.12: Number of OBUs with all content

Strategy	Average (h)	C.I. (95%)
LRBF	1.36	±0.14
LRGF	1.40	±0.14
FILTER	2.49	±0.21

Table 5.5: Statistics of Number of Vehicles with Full Content.

As derived from the Progress Rate, since vehicles receive 100% of the information at a much lower rate, when compared to other strategies, once packets have a certain utility value associated to them, some packets inevitably end up never being a priority during file transmission. As such, a much lower amount of vehicles end up achieving 100% of the information.

Number of Advertisements

Advertisement packets are the main source of study for this Dissertation. As such, the same metrics previously used for them are also present in the Non-Rush Hour timeframe. Figures 5.13 and 5.14 provide to the reader the number of advertisement packets present in the network during this timeframe.

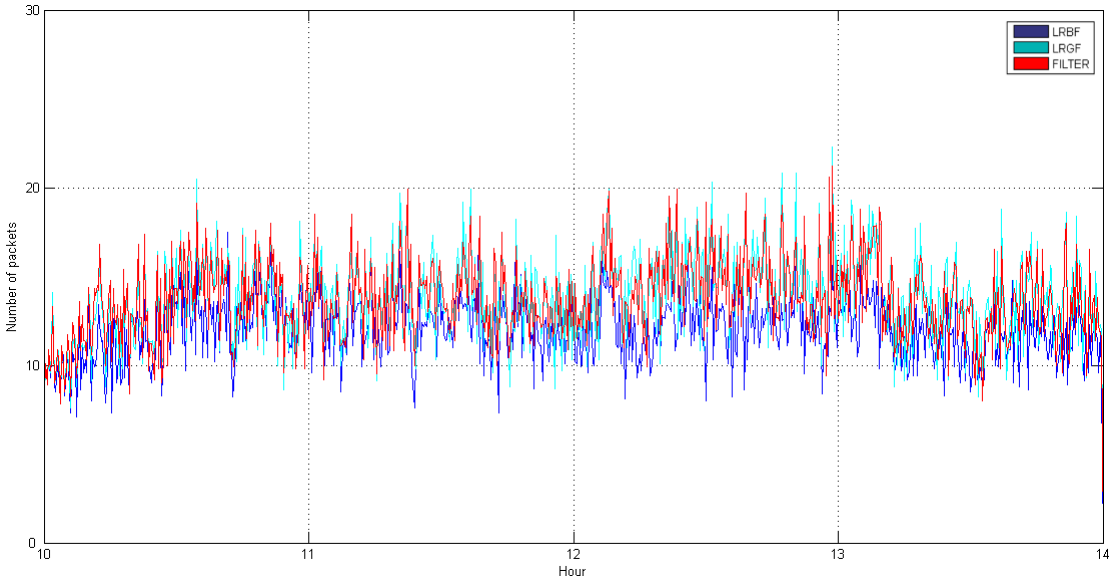


Figure 5.13: Number of transmitted advertisement packets (per timestamp)

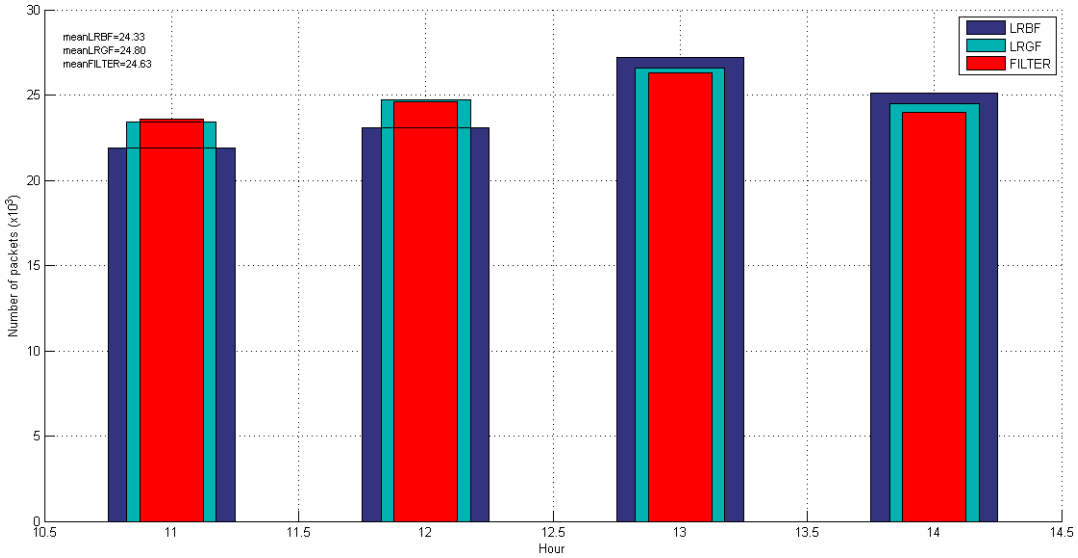


Figure 5.14: Number of transmitted advertisement packets (per hour)

It is evidenced that the number of advertisement packets remains quite similar between all three strategies. However, the small decrease in vehicles does have a clear effect on this dissemination, since the average number of advertisements now rounds 24 thousand packets as opposed to the Rush Hour's 28 thousand packets. Such is not unexpected since the slight decrease in vehicle density also implies a slight decrease in contacts with both the infrastructure and other neighboring vehicles.

Size of Advertisements

This is the metric that evidences the biggest impact on using Bloom Filters to disseminate control packets in the network. Figures 5.15 and 5.16 emphasize this fact by highlighting the size of the transmitted advertisement packets in the network, during this timeframe.

When analyzing the results obtained from the Non-Rush Hour experiment and comparing them with the Rush-Hour one, it is clear that the behaviour of the whole network is highly similar when transmitting advertisement packets. Figure 5.15 shows that during the first two hours, LRBF shows yet again a larger overhead in the remaining two hours, due to most of the dissemination process occurring in the first half of the experiment. Nevertheless, it is slightly smaller since not a single timestamp sees over 120kbps of advertisement files transmitted. FILTER, on the other hand, remains constant in advertisement packets' size since the payload of such packets will always contain bits codifying the storage contents of a node. Such leads to lower network overload. Likewise, LRGF aso evidences good practice in lowering overhead, although its advertisement packets are larger than FILTER's.

Figure 5.9 further corroborates this statement. During the first two hours, LRBF reaches up to 50 MB of network overhead. Albeit smaller than during the Rush Hour experiment, it has to be taken into account that the number of vehicles is also slightly smaller. Its advertising nature makes it so that most nodes rarely send advertisements due to having received the contents of its neighbours. The average size of its advertisement packets reaches approximately 26-27 MB.

LRGF's attempt at lowering overhead proves successful. While following the same behaviour as LRBF, the average size of the overhead throughout the experiment is approximately 18 MB.

FILTER still evidences that the size of the advertisement packets increases over time, which is logical since the progress rate showed that most of the dissemination process happens in the latter half of the experiment. The size of the advertisement packets, however, is still smaller than LRBF and LRGF, reaching an approximate 12 MB, which is still less than half of LRBF's overhead and a third less of LRGF's.

Furthermore, given that the exact same experiment was run multiple times, the variation in both the number and size of sent advertisement packets is minimal, as seen in Table 5.6.

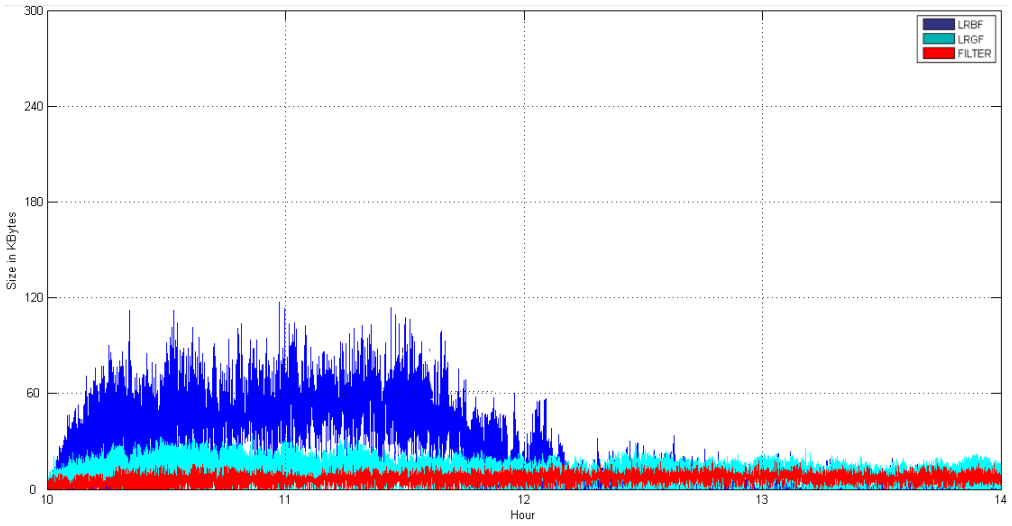


Figure 5.15: Size of transmitted advertisement packets (per timestamp)

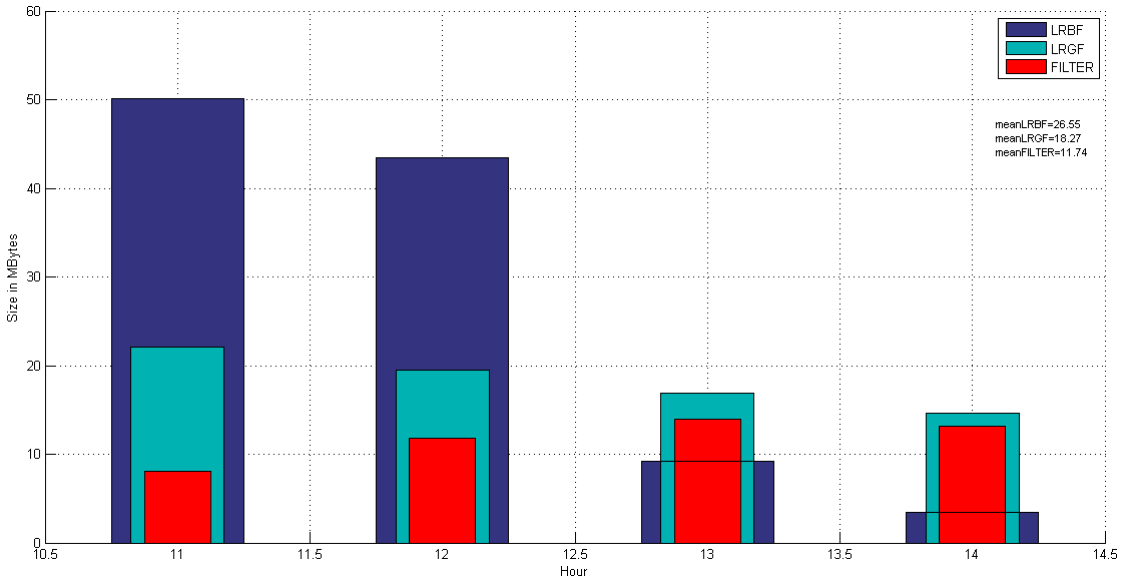


Figure 5.16: Size of transmitted advertisement packets (per hour)

FILTER Strategy	C.I. (95%)
Number	±0.04
Size	±0.02

Table 5.6: Non-Rush Hour Advertisement Packet Statistics.

Server Performance Metrics

Finally, Table 5.7 resumes the physical performance of the server that runs the mOVERS emulator when simulating the vehicular network during the Non-Rush Hour timeframe. The same three metrics are presented to ascertain the machine's correct behaviour.

Strategy	CPU(%)	CPU std(%)	Load	Load std	RAM(%)	RAM std(%)
LRBF	6.0	4.5	1.1	0.9	41.0	7.6
LRGF	12.5	5.1	2.1	0.4	51.2	2.3
FILTER	24.5	8.4	4.5	1.5	33.5	5.9

Table 5.7: Computational Performance in Non Rush Hour.

By comparison with the Rush Hour, all content distribution strategies ran properly in the mOVERS emulator, with no overconsumption of the running machine's resources. Comparing the two strategies themselves, the previously established correlation maintains. FILTER consumes more CPU and machine cores than LRBF and LRGF, once again, due to the nature of the algorithm, such as, the heavy hashing operations derived from it. However, it uses less RAM memory since it does not require constant access to internal structures to gather detailed information about the packets. Instead, to fill the Bloom Filter, it merely uses their identifier, therefore, leading to lower memory consumption. Once again, the increase in resource consumption is a trade-off to ensure lower overhead present in the network.

5.6 Summary

This chapter's focus was on the analysis of the obtained results and posterior evaluation of the same, once the implementation, detailed in Chapter 4, was finalized in mOVERS Emulator. In order to evaluate the behaviour of the vehicular network when performing a content distribution service using Bloom Filters, the Bloom Filter itself was subjected to a test in the time period of most vehicular density to verify its robustness. Furthermore, a previously implemented strategy was used for comparison purposes and tests were run in different time periods to perform such evaluation.

The results obtained in both the Rush Hour and Non-Rush Hour tests have proven that LRBF is still a strategy unmatched in delivery rate, thanks to the intelligence built in its routing algorithm that provides all vehicles with full knowledge of its neighbours' information in storage. However, FILTER has proven a strategy capable of delivering closely the same percentage of information to the network while, at the same time, decreasing the network's overhead considerably.

As such, the use of a Bloom Filter in a vehicular network has proven its benefits and should be a strategy to further optimize as future work.

Chapter 6

Conclusions and Future Work

6.1 Conclusions

The unique characteristics of VANETs such as intermittent connectivity and high mobility make it a network suitable for DTN mechanisms, specifically, due to their SCF (Store-Carry-and-Forward) mechanism trait. The potential of VANETs for non-urgent content distribution (tourist information, multimedia files, among others), while maintaining a low network overhead was the main motivation factor for the development of this Dissertation.

The main aim of this Dissertation was the study of a content distribution strategy with potential for decreasing a vehicular network's overhead, its implementation in the chosen platform for development and posterior evaluation of the obtained results. Several content distribution strategies were studied, specially several using a data structure considered optimal for summarizing content, named *Bloom Filter*. As such, a strategy named FILTER was developed for mOVERS emulator.

This strategy was defined based on pre-existing literature. Bloom Filters have been used several times in vehicular networks. FILTER is a strategy that uses a Bloom Filter to advertise a vehicle's storage content by encapsulating it within a control packet and broadcasting it to all of a vehicle's neighbours. This control packet adds overhead to the network since it is not the actual content required by the vehicles.

Several challenges arose during the development of this Dissertation, most of which were related to the analysis of the emulator's code in order to properly understand all the procedures when running a mOVERS test as well as to the implementation of the strategy itself.

In order to test the behaviour of the proposed content distribution strategy, the mOVERS emulator was used to implement the code required to test such strategy. Two preferred scenarios were chosen based on pre-existing literature in order to evaluate the impact of the Bloom Filter-based strategy on the vehicular network of Oporto: the Rush Hour and the Non-Rush Hour.

After evaluating the data obtained from the tests, several conclusions were derived. LRBF provides the upper bound for delivery ratio, since the nodes will have knowledge of the complete information of all the packets in each neighbouring node. However, the high amount of

information transmitted in the control packets cause the network to have a considerably high network overhead. FILTER, the strategy chosen in the scope of this Dissertation, has proven to be able to attenuate such network overhead by sending storage content information of each vehicle, coded into bits, leading to a lower control packet size and ultimately, lower network overhead. This strategy only failed in fully mimicking the delivery rate of LRBF by roughly 2%. Such must be seen as the trade-off to lower the network's overhead.

It was concluded that, in order to attempt to improve the delivery rate of this strategy, the function used to re-order the storage content (labeled Utility Function) should be further improved to ensure a faster packet distribution in the first half of the tests run, since most of the dissemination with the proposed strategy occurs in the latter half.

6.2 Future Work

The field of Content Distribution has not been fully exploited yet and, as such, has yet many topics worthy of evaluation. Future studies in this area should include the following improvements:

- **mOVERS Emulator Performance:** The emulator used can be highly unstable when attempting to perform a test that requires a high number of nodes in a highly dense network. Moreover, the emulator is run on a server that requires resource sharing, leading to a potential abrupt shutdown. Given that mOVERS requires a high amount of time to emulate a given number of nodes that may require further decrease, the reason for this instability should be assessed in the code and solved for emulator robustness.

- **Standard Filter Optimization:** Previous content distribution strategies ensured a decrease in overhead during the experiment by making the vehicles announce their fully completed storage as a shorter message. The current implementation of the filter always builds a bit table with the number of packets coded into it. A future optimization would be making the nodes announce their fully completed storage by sending a shorter message, as well.

- **New Filter Types:** In the scope of this Dissertation, a standard filter was used to test the network for decreased overhead. However, other types of filters could be used to further optimize the network overhead and add extra functionalities to the network, such as tracking the presence of specific packets in nodes through time.

- **Utility Function Study:** A study on several utility functions should be performed to assess the most suitable method to ensure enhanced packet distribution and, consequently, an enhanced delivery rate when transmission is validated by the control packets.

- **Real World Evaluation:** Both the proposed and the previously implemented content distribution strategies have only been tested in the emulator with data from the vehicular network mobility. In the future, tests in the real equipments, OBUs and RSUs, will

be performed to evaluate the strategies' performance on a real life network. This will require the compilation of mOVERS to the boards within a controlled laboratory environment, and then a real evaluation in the real 'uncontrolled' network.

Bibliography

[1] D. Evans. The internet of things - how the next evolution of the internet is changing everything. *http://www.cisco.com/c/dam/en_us/about/ac79/docs/innov/IoT_IBSG_0411FINAL.pdf*, 2011.

[2] G. Scuro. Automotive industry: Innovation driven by electronics. *http://embedded-computing.com/articles/automotive-industry-innovation-driven-electronics/*, 2013.

[3] T. Klier. Making cars smarter: The growing role of electronics in automobiles. *http://www.chicagofed.org/digital_assets/publications/chicago_fed_letter/2011/cfloctober2011_291a.pdf*, 2011.

[4] Veniam. An internet of moving things. *http://veniam.com*, 2016.

[5] K. Fall. A delay-tolerant network architecture for challenged internets. *Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*, 2003.

[6] H. Moustafa and Y. Zhang. Vehicular networks: Techniques, standards and applications. *Chapter: Special Characteristics of Vehicular Networks*, 2009.

[7] R. Kumar and M. Dave. A review of various vanet data dissemination protocols. *International Journal of u- and e- service*, 2012.

[8] S. Al-Sultan. A comprehensive survey on vehicular ad hoc network. *Journal of Network and Computer Applications archive, Volume 37, pages 380-392*, 2014.

[9] IEEE. Ieee 1609 - family of standards for wireless access in vehicular environments (wave). *https://www.standards.its.dot.gov/factsheets/factsheet/80*, 2009.

[10] D. Jiang. Ieee 802.11p: Towards an international standard for wireless access in vehicular environments. 2008.

[11] A. Ghandour. Dissemination of safety messages in ieee 802.11p/wave vehicular network: Analytical study and protocol enhancements. *Pervasive and Mobile Computing, Volume 11, pages 3-18*, 2014.

[12] D. Carreira. Mobility of communications between vehicles and infrastructure. *Ms.C Dissertation, Universidade de Aveiro*, 2013.

[13] NASA. Disruption tolerant networking. *https://www.nasa.gov/content/dtn*.

[14] Internet Research Task Force. Delay-tolerant networking architecture. *Draft RFC-4838*.

[15] F. Warthman *et al.* Delay and disruption-tolerant networks (dtns), a tutorial. *Interplanetary Internet Special Interest Group*.

[16] G. Pessoa. Content distribution in vehicular networks using delay-tolerant communication mechanisms. *MsC Dissertation, Instituto de Telecomunicações, Universidade de Aveiro*, 2015.

[17] P. Pereira. From delay-tolerant networks to vehicular delay-tolerant networks. *IEEE Communications Surveys & Tutorials*, 2012.

[18] S. Zeadally. Vehicular ad hoc networks (vanets): Status, results, and challenges. 2012.

[19] A. Festag. Now – network on wheels': Project objectives, technology and achievements. *5th International Workshop on Intelligent Transportation (WIT), pages 211-216*, 2008.

[20] PreVENT. Preventive and active safety application. *http://www.transport-research.info/project/preventive-and-active-safety-application*, 2008.

[21] S. Guo. Very low-cost internet access using kiosknet. *https://www.researchgate.net/publication/228646337_Very_low-cost_Internet_access_using_KioskNet*, 2012.

[22] T. Gruber. Coopers - co-operative systems for intelligent road safety. *http://www.ait.ac.at/research-services/research-services-digital-safety-security/verification-and-validation/reference-projects/projects-completed/coopers-co-operative-systems-for-intelligent-road-safety/?L=1*, 2016.

[23] B. Hull. Cartel: A distributed mobile sensor computing system. *Proceedings of the 4th international conference on Embedded networked sensor systems, pages 125-138*, 2006.

[24] J. Ott. A disconnection-tolerant transport for drive-thru internet environments. *Proceedings IEEE for the 24th Annual Joint Conference of the IEEE Computer and Communications Societies*, 2005.

[25] H. Soroush. Dome: A diverse outdoor mobile testbed. *Proceedings of the 1st ACM International Workshop on Hot Topics of Planet-Scale Mobility Measurements*, 2009.

[26] S. Lee. Content distribution in vanets using network coding: The effect of disk i/o and processing o/h. *5th Annual IEEE Communications Society Conference on Sensor, Mesh and Ad Hoc Communications and Networks*, 2008.

[27] M. Gerla. Content distribution in vanets. *Vehicular Communications, Volume 1, Issue 1, pages 3-12*, 2014.

[28] S. Das. Spawn: A swarming protocol for vehicular ad-hoc wireless networks. *Vehicular Communications*, 2004.

[29] K. Lee. Cartorrent : A bit-torrent system for vehicular ad-hoc networks. *University of California, Los Angeles*, 2006.

[30] Y. Zhang. Roadcast: A popularity aware content sharing scheme in vanets. *ACM SIGMOBILE Mobile Computing and Communications Review*, 2010.

[31] H. Wu. Mobility-centric data dissemination algorithm for vehicular networks. *Proceedings of the 1st ACM International Workshop on Vehicular ad hoc networks*, 2004.

[32] B. Yu. Pyramid: Informed content reconciliation for vehicular peer-to-peer systems. *IEEE Vehicular Networking Conference*, 2015.

[33] A. Marandi. Practical bloom filter based epidemic forwarding and congestion control in dtns: A comparative analysis. *Computer Communications, Volume 48, pages 98-110*, 2014.

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

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

[36] E. Yoneki. An adaptive approach to content-based subscription in mobile ad hoc networks. *Proceedings of the Second IEEE Annual Conference on Pervasive Computing and Communications Workshops*, 2004.

[37] I. Parris. Privacy-enhanced social network routing in opportunistic networks. *8th IEEE International Conference on Pervasive Computing and Communications Workshops*, 2010.

[38] F. Angius. Bloogo: Bloom filter based gossip algorithm for wireless ndn. *Proceedings of the 1st ACM workshop on Emerging Name-Oriented Mobile Networking Design - Architecture, Algorithms, and Applications*, 2012.

[39] A. Hassan. Vanet simulation. *Ms.C Dissertation, Halmstad University*, 2009.

[40] D. Zier. Netsim: An object-oriented architectural simulator suite. *Proceedings of the 2005 International Conference on Computer Design*, 2005.

[41] A. Tomandl. Vanetsim: An open source simulator for security and privacy concepts in vanets. *International Conference on High Performance Computing and Simulation*, 2014.

[42] SUMO. Sumo - simulation of urban mobility. *http://www.dlr.de/ts/en/desktopdefault.aspx/tabid-9883/16931_read-41000/*.

[43] M. Piorkowski. Trans: Realistic joint traffic and network simulator for vanets. *ACM SIGMOBILE Mobile Computing and Communications Review 12*, 2008.

[44] S. Jain. Routing in a delay tolerant network. *Proceedings of the 2004 conference on Applications, technologies, architectures, and protocols for computer communications*, 2004.

[45] A. Keranen. The one simulator for dtn protocol evaluation. *Proceedings of the 2nd International Conference on Simulation Tools and Techniques*, 2009.

[46] T. Condeixa. Delay-tolerant networking. *Veniam Presentation*, 2014.

[47] P. Basu. Opportunistic forwarding in wireless networks with duty cycling. *Proceedings of the third ACM workshop on Challenged networks, pages 19-26*, 2008.

[48] A. Vahdat. Epidemic routing for partially-connected ad hoc networks. *Technical Report, Duke University*, 2000.

[49] A. Broder. Network applications of bloom filters: A survey. *Internet Mathematics Vol. I*, 2004.

[50] J. Trindade. An efficient overlay guiding system for routing control messages. *Ph.D Dissertation, Instituto Superior Técnico - Universidade Técnica de Lisboa*, 2011.

[51] M. Mitzenmacher. Compressed bloom filters. *IEEE/ACM Transactions on Networking, Volume 10, pages 604-612*, 2002.

[52] L. Fan. Summary cache: A scalable wide-area web cache sharing protocol. *IEEE/ACM Transactions on Networking, Volume 8, pages 281-293*, 2000.

[53] S. Rhea. Probabilistic location and routing. *INFOCOM 2002. Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings.*, 2002.

[54] B. Maggs. Algorithmic nuggets in content delivery. *ACM SIGCOMM Computer Communication Review, Volume 45, pages 52-66*, 2016.

[55] F. Mezghani. Utility-based forwarder selection for content dissemination in vehicular networks. *IEEE 25th International Symposium on Personal, Indoor and Mobile Radio Communications*, 2014.

[56] L. Wischhof. Congestion control in vehicular ad-hoc networks. *IEEE International Conference on Vehicular Electronics and Safety*, 2005.

[57] A. Partow. General purpose hash function algorithms. *http://www.partow.net/programming/hashfunctions/*.

[58] Porto Digital. Porto digital - o projecto. *https://portodigital.pt/index.php?artigo=1*, 2014.

[59] GoogleMaps. Oporto's vehicular network. *https://www.google.pt/maps*.

Appendix A

How to run tests on mOVERS Emulator

The following appendix serves as a guide on how to run a DTN scenario test in the mOVERS emulator directly in a machine.

A.1 Compiling

Once within the folder of the mOVERS emulator, change the location to **libhelix/src** and compile the mOVERS' libraries with the command **sudo make clean all install**. This command does not need to be run every time a test is run. Only whenever the libraries are changed, for example, if new members are added to the header of the packets (information located in the libraries).

Afterwards change the location to **mOVERS/build** and generate a Makefile for compiling, using the command **sudo cmake -DEMULATOR=ON ../src** and compile using **sudo make clean all emugui**. This will generate binary files in the folder **mOVERS/bin** that must be then moved to the server that will be running the emulator. It is not advisable to run this emulator in a personal laptop due to the heavy consumption of resources inherent to this emulator.

A.2 Execution and Monitoring

The mOVERS execution is done using the script *startEMU*. However, some fields within will have to be changed depending on the test being run.

- **Dataset**: The field for choosing which mobility pattern to analyze. Choice is either 2 or 3, standing for the dataset collected in October 31st, 2014 or February 13th, 2015;
- **Initial and Final Time**: The timeframe must be set by changing the fields **ti** and **tf** to the corresponding UNIX timestamps.

During mOVERS execution, the behaviour of the OBUs and RSUs can be monitored by the user by changing the location to the folder **mOVERS/scripts/nohup_files**. The *nohup_-emugui.out* allows the user to confirm the correct timeframe of execution set by the user.

Moreover, each OBU and RSU generates an *.out* file in this location where the user can choose to "tail" a file to monitor the machine's behaviour during the experiment, using the command **tail -f nohup_**u_id.out**, where "**u" can be an obu or rsu and the id is the corresponding number of the machine.

A.3 Logging

The files for posterior analysis in MATLAB can be found in **mOVERS/logs/datasetx** where x stands for the chosen dataset (2 or 3). Those files must then be transferred to the machine running MATLAB.