Universidade de Aveiro Departamento de Electrónica, Telecomunicações e Informática,
2015

Gonçalo da Silva Pessoa

Distribuição de Conteúdos em Redes Veiculares usando Mecanismos de Comunicação Tolerantes ao Atraso

Content Distribution in Vehicular Networks using Delay-Tolerant Communication Mechanisms

Universidade de Aveiro Departamento de
Electrónica, Telecomunicações e Informática,
2015

Gonçalo da Silva Pessoa

Distribuição de Conteúdos em Redes Veiculares usando Mecanismos de Comunicação Tolerantes ao Atraso

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requesitos necessários à obtenção do grau de Mestre em Engenharia Eletrónica e Telecomunicações, realizada sob a orientação científica da Professora Doutora Susana Sargento, Professora Associada com Agregação do Departamento de Eletrónica Telecomunicações e Informática da Universidade de Aveiro e do Doutor Ricardo Matos, Systems Engineer na Veniam.

o júri / the jury

presidente / president	**Professor Doutor Rui Luís Andrade Aguiar** Professor Catedrático da Universidade de Aveiro
vogais / examiners committee	**Professora Doutora Susana Isabel Barreto de Miranda Sargento** Professora Associada com Agregação da Universidade de Aveiro (orientadora)
	Professor Doutor Augusto José Venâncio Neto Professor Adjunto da Universidade Federal do Ceará

Acknowledgments

The writing of this dissertation has been one of the most significant and fulfilling experiences I have ever had, both academically and personally. There are several people who, directly or indirectly, have contributed to the successful completion of this endeavor and without whose support, patience and guidance this journey would have not been the same. To them, I owe my appreciation.

First and foremost, to my supervisor for her invaluable guidance and comments and for always asking for "that extra mile". She was instrumental.

To my parents, for their unconditional love, support and inestimable advice. For never letting me lose sight of my objectives.

To Filipa, for her friendship, presence, patience, guidance, and love in these last 7 years. Without you I would not be here, nor would I be the person that I am.

To Tiago, who was a fundamental support in all this process, his guidance and companionship were priceless to develop this work and overcome all of the challenges.

To my dear colleagues Marco, André, Gonçalo, Bojan, and João for their support and friendship during these hard but pleasant years.

To my best and most important friends, the family I chose and that forces me to have fun when I forget to.

Resumo

Nas últimas décadas tem-se assistido à introdução de novas redes de telecomunicações. Entre estas destacam-se as redes veiculares constituídas por todo o tipo de veículos com capacidades de intercomunicação.

As redes veiculares têm especificidades singulares face a outro tipo de redes devido à constante mobilidade dos nós e à sua elevada dispersão geográfica. Os principais desafios introduzidos por este tipo de redes prendem-se com a conectividade intermitente e o atraso longo e variado na entrega da informação.

Por forma a fazer face aos problemas relacionados com a conectividade intermitente, introduziu-se um novo conceito intitulado de Delay Tolerant Network (DTN). Esta arquitetura assenta num mecanismo de *Store-Carry-and-Forward (SCF)* por forma a garantir a entrega de informação em situações onde não existe um caminho estabelecido fim-a-fim.

As redes veiculares suportam uma multiplicidade de serviços, nos quais se inclui o transporte de informação não-urgente. Desta forma, a utilização de uma DTN para a difusão de informação não-urgente permite ultrapassar os desafios identificados anteriormente.

O trabalho realizado foca-se na utilização de DTNs para a disseminação de informação não-urgente. Por forma a operacionalizar esta premissa foram implementadas quatro estratégias distintas: Random, Least Number of Hops First (LNHF), Local Rarest Bundle First (LRBF) e Local Rarest Generation First (LRGF). Todas estas estratégias tem um objectivo comum: disseminar um conteúdo na rede no menor tempo possível minimizando ao máximo o congestionamento da rede. Foram também implementadas e estudadas técnicas para minimizar o congestionamento do meio.

A metodologia de desenho, implementação e validação das estratégias propostas foi desenvolvida em três fases. A primeira focou-se na criação de um emulador Matlab para a implementação rápida e validação das estratégias. Dessa primeira fase resultaram quatro estratégias que foram posteriormente implementadas no software de DTNs Helix desenvolvido através de uma parceria entre o Instituto de Telecomunicações (IT) e a Veniam® (responsáveis pela maior rede veicular em operação a nível mundial localizada na cidade do Porto). As estratégias foram depois avaliadas num emulador construído para fazer testes de grande escala. Ambos os emuladores introduzem a mobilidade dos veículos com base em informação recolhida previamente da plataforma real. Por fim a estratégia que apresentou o melhor desempenho foi introduzida e testada numa plataforma real para demonstração de conceito e operacionalidade.

Conclui-se que duas das estratégias implementadas (LRBF and LRGF) são passíveis de utilização na rede real garantido uma taxa de entrega significativa. A estratégia LRBF apresentou o melhor desempenho em termos de entrega, no entanto, necessita de adicionar um overhead considerável na rede para funcionar. No futuro devem ser realizados testes de escalabilidade em ambiente real por forma a confirmar os resultados obtidos em ambiente de emulação e real em pequena escala. A implementação real das estratégias deve ser acompanhada pela introdução de novos tipos de serviços para distribuição de conteúdos.

Abstract

The last couple of decades have been the stage for the introduction of new telecommunication networks. It is expected that in the future all types of vehicles, such as cars, buses and trucks have the ability to intercommunicate and form a vehicular network.

Vehicular networks display particularities when compared to other networks due to their continuous node mobility and their wide geographical dispersion, leading to a permanent network fragmentation. Therefore, the main challenges that this type of network entails relate to the intermittent connectivity and the long and variable delay in information delivery.

To address the problems related to the intermittent connectivity, a new concept was introduced – Delay Tolerant Network (DTN). This architecture is built on a Store-Carry-and-Forward (SCF) mechanism in order to assure the delivery of information when there is no *end-to-end* path defined.

Vehicular networks support a multiplicity of services, including the transportation of non-urgent information. Therefore, it is possible to conclude that the use of a DTN for the dissemination of non-urgent information is able to surpass the aforementioned challenges.

The work developed focused on the use of DTNs for the dissemination of non-urgent information. This information is originated in the network service provider and should be available on mobile network terminals during a limited period of time. In order to do so, four different strategies were deployed: Random, Least Number of Hops First (LNHF), Local Rarest Bundle First (LRBF) e Local Rarest Generation First (LRGF). All of these strategies have a common goal: to disseminate content into the network in the shortest period of time and minimizing network congestion. This work also contemplates the analysis and implementation of techniques that reduce network congestion.

The design, implementation and validation of the proposed strategies was divided into three stages. The first stage focused on creating a Matlab emulator for the fast implementation and strategy validation. This stage resulted in the four strategies that were afterwards implemented in the DTNs software Helix – developed in a partnership between Instituto de Telecomunicações (IT) and Veniam®, which are responsible for the largest operating vehicular network worldwide that is located in Oporto city. The strategies were later evaluated on an emulator that was built for the large-scale testing of DTN. Both emulators account for vehicular mobility based on information previously collected from the real platform. Finally, the strategy that presented the best overall performance was tested on a real platform – in a lab environment – for concept and operability demonstration.

It is possible to conclude that two of the implemented strategies (LRBF and LRGF) can be deployed in the real network and guarantee a significant delivery rate. The LRBF strategy has the best performance in terms of delivery. However, it needs to add a significant overhead to the network in order to work. In the future, tests of scalability should be conducted in a real environment in order to confirm the emulator results. The real implementation of the strategies should be accompanied by the introduction of new types of services for content distribution.

Contents

List of Figures

List of Tables

List of Equations

List of Algorithms

Acronyms

ACK	Acknowledgment
ADU	Application Data Unit
AODV	Ad hoc On-demand Distance Vector
AP	Access Point
API	Application Programming Interface
AU	Application Unit
BAD	Bundle Aggregation and De-aggregation
BER	Bit Error Rate
BPA	Bundle Protocol Agent
BSC	Bundle Signaling Control
BSP	Bundle Security Protocol
BSS	Basic Service Set
C2C	Car-to-Car
C2C-CC	Car-to-Car Communication Consortium
CAN	Controller Area Network
CanuMobiSim	CANU Mobility Simulation Environment
CBHE	Compressed Bundle Header Encoding
CCH	Control Channel
CGR	Contract Graph Routing
CLA	Convergence Layer Adapter
CPU	Central Processing Unit
CSV	Comma-Separated Values
CVS-VN	Cooperative Video Streaming over Vehicular Networks

DARPA	Defense Advanced Research Projects Agency
DIVERT	Development of Inter-VEhicular Reliable Telematics
DOME	Diverse Outdoor Mobile Environment
DSRC	Dedicated Short Range Communications
DTLSR	Delay Tolerant Routing for Developing Regions
DTN	Delay Tolerant Network
DTNRG	Delay Tolerant Network Research Group
E2E	End-to-End
EC	European Commission
EID	Endpoint Identifier
HTTP	Hyper Text Transfer Protocol
HTTPS	Hyper Text Transfer Protocol Secure
FCC	Federal Communications Commission
GSM	Global System for Mobile Communications
GPS	Global Positioning System
GUI	Graphical User Interface
IBR	Institut für Betriebssysteme und Rechnerverbund
iCS	iTETRIS Control System
IEEE	Institute of Electrical and Electronics Engineers
IETF	Internet Engineering Task Force
ION	Interplanetary Overlay Network
IP	Internet Protocol
IPC	Inter-Process Communication
IPND	IP Neighbor Discovery
ISTEA	Intermodal Surface Transportation Efficiency Act
IVC	Inter-Vehicle Communication
IVHS	Intelligent Vehicle/Highway System
IT	Instituto de Telecomunicações
ITS	Intelligent Transportation Systems

ITSA	Intelligent Transportation Society of America
JPL	Jet Propulsion Laboratory
JSON	JavaScript Object Notation
LNHF	Least Number of Hops First
LRBF	Local Rarest Bundle First
LRGF	Local Rarest Generation First
LTE	Long Term Evolution
LTP	Licklider Transmission Protocol
MAC	Medium Access Control
MANET	Mobile Ad-Hoc NETwork
MATLAB	MATrix LABoratory
MDDV	Mobility-Centric Data Dissemination
MGW	Mobile GateWay
MLME	MAC Layer Management Entity
MULE	Mobile Uniquitous LAN Extensions
NAP	Network Architectures and Protocols
NCTUns	National Chiao Tung University Network Simulator
NITSA	National Intelligent Transportation Systems Architecture
NTP	Network Time Protocol
OBS	Optical Burst Switching
OBU	On-Board Unit
ONE	Opportunistic Network Environment
OS	Operating System
OSI	Open Systems Interconnection
P2P	Peer-to-Peer
PAN	Personal Area Network
PARAMICS	PARAllel MICroscopic Simulation of road traffic
PHP	Hypertext Preprocessor
PHY	Physical

PLME	Physical Layer Management Entity
PRoPHET	Probabilistic Routing Protocol using History of Encounters and Transitivity
PS	Provider Service
PTPd	Precision Time Protocol daemon
QoS	Quality of Service
RAM	Random Access Memory
REDEC	REceiver-based solution with video transmission DECoupled
RFC	Request for Comments
RLNC	Random Linear Network Coding
RSSI	Received Signal Strength Indicator
RSU	Road Side Unit
RTEM	Real-Time Executive for Multiprocessor System
SAE	Simultaneous Authentication of Equals
SaW	Spray-and-Wait
SCF	Store-Carry-and-Forward
SCH	Service Channel
SDR	Simple Data Record
SPAWN	Swarming Protocol For Vehicular Ad-Hoc Wireless Networks
SQL	Structured Query Language
STA	Collection Station
STCP	Sociedade de Transportes Coletivos do Porto
SUMO	Simulation Urban MObility
TCA	Tetherless Computing Architecture
TCL	Tool Command Language
TCP	Transmission Control Protocol
TDMA	Time Division Multiple Access
TLS	Transport Layer Security
TraCI	Traffic Control Interface

TraNS	Traffic and Network Simulation Environment
TUB	Technische Universität Braunschweig
UDP	User Datagram Protocol
UDS	Unix Domain Socket
UMTS	Universal Mobile Telecommunications System
URI	Uniform Resource Identifier
US	United States
USDOT	United States Department of Transportation
UTC	Coordinated Universal Time
UWME	Universal WAVE Management Entity
V2I	Vehicle-to-Infrastructure
V2V	Vehicle-to-Vehicle
VANET	Vehicular Ad-hoc NETwork
VanetMobiSim	VANET Mobility Simulation Environment
VDTN	Vehicular Delay Tolerant Network
VEINS	VEhicles In Network Simulation
VINT	Virtual InterNetwork Testbed
VM	Virtual Machine
WAVE	Wireless Access in Vehicular Environments
Wi-Fi	Wireless Fidelity
WME	WAVE Management Entity
WSMP	Wave Short Messaging Protocol
WSN	Wireless Sensor Network
XML	eXtensible Markup Language
ZMQ	Zero-M Queue
ZoR	Zone-of-Relevance

Chapter 1

Introduction

This document was developed under the scope of the Dissertation of the Master in Electronics and Telecommunications Engineering in the Departamento de Eletrónica, Telecomunicações e Informática of Universidade de Aveiro with the theme of "delay-tolerant communication mechanisms for vehicular networks".

In this chapter the context and motivation of this Dissertation is presented as well as its main objectives and contributions. It also gives a brief description of the document organization.

1.1 Context and Motivation

The last couple of decades have been the stage for the introduction of new telecommunication networks. It is expected that, in the future, all types of vehicles, such as cars, buses and trucks have the ability to intercommunicate and form a vehicular network. A cooperation between the Universities of Aveiro and Porto, Instituto de Telecomunicações (IT), and a spin-off of both of them, Veniam® [1], lead to the implementation of the largest – on a global level – vehicular communication platform. This platform comprises more than 600 vehicles (taxis, trucks and buses) and fixed stations and is located in Oporto city.

Vehicular networks display special characteristics when compared to other networks due to their continuous node mobility and their wide geographical dispersion. Therefore, the main challenges that this type of networks entails relate to the intermittent connectivity and the long and variable delay in information delivery due to permanent network fragmentation and node mobility.

In a completely distinct context than that of vehicular networks, a new architecture was suggested with the goal of allowing the reliable transmission of information in the scope of high network fragmentation (intermittent connectivity) and high latency between contacts. Thus, a new concept was presented – Delay Tolerant Network (DTN) [2]. This architecture introduces the Store-Carry-and-Forward (SCF) mechanism in order to assure information delivery when there is no end-to-end path defined.

The aforementioned challenges limit the spectrum of possible vehicular network services and applications. Services that involve low transmission latencies and a stable path between the sender and the receiver face many challenges and need to resort to different technologies (e.g. cellular networks) that typically imply a higher service price. Bearing these limitations in mind, vehicular networks can support a variety of services, such as Internet access, emergency services, or delivery of sensory information. Taking into consideration this array of services, these can be divided into two categories, considering the kind of information that is being delivered: urgent and non-urgent information. Therefore, it is possible to conclude that the use of a DTN for non-urgent information dissemination is able to surpass the aforementioned challenges.

This is the framework in which this dissertation is developed. This work is focused on using DTNs for the dissemination of non-urgent information, such as advertisement videos, tourism-related information (not interactive), movies and tv-shows (not in real-time), among others, that should be made available in the mobile terminals of the network (taxis and buses) during a limited period in time and that originate in the network service provider.

Alongside this dissertation a specific DTNs solution (Helix) was designed by IT and Veniam® in order to be implemented in the aforementioned vehicular network. However, Helix was mainly focused on the functional structure of the architecture supporting a delay-tolerant network and how it should be adapted to the particularities of a vehicular context. The routing module of this software, responsible for the decision to send and receive information, was only able to collect data from passing vehicles directly to the network fixed infrastructure. Thus, this software only supported the direct upload of information from mobile nodes to the fixed one, not using the vehicular network to transport or spread data in broadcast through vehicles. This module is an important part for the introduction of any kind of new services regarding the spreading or sharing of information between the network nodes, and its constrains limit the introduction of new types of services, such as non-real-time content distribution. As such, the optimization and evolution of the routing module is imperative to create the necessary conditions for the introduction of these services. Moreover, other modifications and implementations must be performed in order to introduce those kinds of services.

Figure 1.1 illustrates a comprehensive view of the envisioned scenario for this Dissertation. This scenario illustrates a process of content dissemination from a remote content server, located in the core network, to mobile nodes (vehicles equiped with an On-Board Unit (OBU)) using the vehicular network to transport the data. The content under dissemination is stored in a set of remote servers which are directly connect to the fixed vehicular network infrastructure, Road Side Units (RSUs), allowing the permanent availability of such content in these fixed nodes. Once available in these fixed nodes, the content can be spread directly to the mobile nodes that pass by them. Those mobile nodes can send the content to their peers in order to reach a maximum delivery rate.

A content distribution strategy has the main goal of spreading the content under dissemination to a number of network nodes as high as possible and minimizing the network resources consumption during this period. Thus, this kind of service needs to be correctly designed and implemented in order to achieve such goals. As illustrated in Figure 1.1, all the sender nodes (fixed and mobile) are performing a routing decision of which packet should be sent (in broadcast) in order to maximize the delivery rate and minimize the network congestion. As a statement, in Figure 1.1, the transmissions are not performed all at the same time. The sender nodes must be able to decide which is the information to be sent, when it should be sent, to what kind of nodes, what is the goal of such content, along any other important parameter or characteristic associated with the content and network. Thus, any proposed solution for a content distribution scenario must address all the previous goals and key factor decisions, in order to be correctly deployed.

It is within this context and with the main goal of studying optimized content distribution

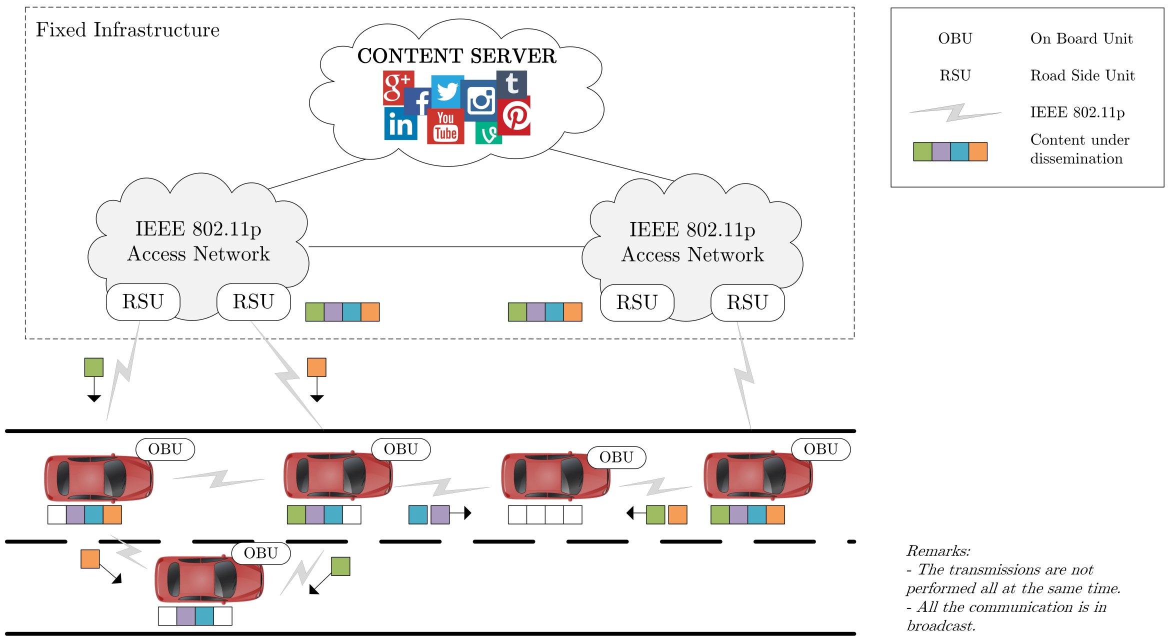

Fixed Infrastructure
CONTENT SERVER
IEEE 802.11p Access Network
IEEE 802.11p Access Network
RSU
RSU
RSU
RSU
OBU
OBU
OBU
OBU
OBU
OBU
On Board Unit
RSU
Road Side Unit
IEEE 802.11p
Content under dissemination
Remarks:
- The transmissions are not performed all at the same time.
- All the communication is in broadcast.

Figure 1.1: Content distribution to-be scenario (illustrative)

strategies for the dissemination of non-urgent information through vehicular networks that this Dissertation appears.

1.2 Objectives and Contributions

The main goal of this dissertation is the study and implementation of strategies to disseminate non-urgent content (e.g. adds, videos, and tourism-related information) through a vehicular network in the smallest time period along with minimizing network congestion and resource usage during this process. With this goal in mind the present Dissertation has the following objectives:

- *Survey of related work*: before starting the implementation and evaluation of new strategies it is necessary to understand the state of the art in this research field.
- *Study the existent platforms of development and evaluation*: several platforms are available to perform the evaluation of routing protocols, although not all of them are suitable for a content distribution experiment using data from a real network as a mobility model and to develop content distribution schemes specifically designed for a dedicated DTN software (Helix).
- *Study of the real vehicular network*: since data from a real vehicular network is used, several analyses can be performed regarding a better understanding of vehicle mobility and network behavior.
- *Design and implement strategies to stateless choose information*: the main goal of this dissertation is the designing and implementation of content distribution strategies which

are closely related with a correct selection of the packets to be broadcasted and the most propitious time to send the information.

- *Design and implement strategies to disseminate information*: since the transmission of contents is in broadcast, several challenges arise such as network congestion that must be controlled and mitigated, whereby it is necessary to design and evaluate strategies to control the dissemination of information.

- *Evaluate the behavior of implemented strategies*: once the strategies are designed and implemented, it is crucial to analyze their behavior and performance.

The answer to the previous objectives and challenges results in several contributions that can be summarized as follows:

- *Development of a content distribution emulator*: the existent emulation platforms were not suitable or appropriate for the design and evaluation of content distribution strategies in a vehicular network based on real log data, whereby during this work two other emulators were designed and implemented. The first one, called MatlabEmulator, was developed specifically to study and demarcate suitable strategies to stateless choose and disseminate information and the other one is specific for developing protocols for the Helix software and evaluate their scalability and performance, the HelixEmulator.

- *Improvement of the HelixEmulator*: given that this emulator was in an early state of stabilization, several bugs were detected and fixed and new features were introduced such as as the logging module and the capability to transmit in broadcast.

- *Development of the Helix content dissemination service support*: this DTN implementation specifically designed for a vehicular environment did not have any feature or module regarding a widespread content distribution service. Thus, during this work several content distribution strategies were designed and implemented to be used in Helix software in order to provide the additional content distribution service. Regarding this goal, several new modules and features (e.g. coding approach, exchange of content advertisement messages, etc.) were added to support this new service.

- *Statistical analysis and understanding of the real vehicular network*: as several logging data was collected from a real vehicular network and the network behavior was studied during this work, several statical analyses were performed and this valuable knowledge can be used in the future.

- *Design and development of several content distribution strategies in multiple platforms*: regarding the study of multiple strategies, several approaches were designed and implemented on multiple platforms resulting in four major strategies that where implemented for Helix software, being a relevant addition to it.

- *Evaluation of several content distribution strategies in multiple platforms*: regarding the evaluation of the proposed content distribution strategies, these were evaluated on three different platforms (two network emulators - large scale, and one real testbed - small-scale) to become this set of strategies more efficient and robust.

- *Real testbed and performance evaluation*: gives a clear demonstration of concept and feasibility of the most suitable content distribution strategy.

As a result of this work a scientific paper entitled "Content distribution in vehicular networks using delay-tolerant networks" was presented in the 10th Conference on Telecommunications (Conftele) 2015, a paper entitled "Emulator for Content Distribution in Vehicular Networks" will be submitted to an IEEE conference, and a journal paper containing the strategies and results of the content distribution process will be submitted to an international journal.

1.3 Document Structure

The present dissertation is structured in seven fundamental chapters as follows:

- *Chapter 1 - Introduction* - this chapter provides the scope and motivation of the dissertation, main objectives and contributions, and the document structure.
- *Chapter 2 - Fundamental Concepts* - this chapter provides to the reader an overview analysis and description of the fundamental concepts related with this dissertation: vehicular networks and delay-tolerant networks.
- *Chapter 3 - Related Work* - this chapter provides a wide survey on content distribution schemes over mobile ad-hoc networks where the mobility of the nodes is a significant challenge along with the network resources constraints such as Central Processing Unit (CPU) and memory usage, network congestion, among others.
- *Chapter 4 - Content Distribution Schemes* - this chapter presents the proposed four content distribution strategies to stateless choose information and a set of techniques to control the dissemination of information.
- *Chapter 5 - Integration and Development* - this chapters describes the design, implementation and integration procedures of the proposed strategies and techniques in the different platforms used for evaluation.
- *Chapter 6 - Evaluation* - this chapter describes the evaluated scenarios, the equipment and platforms used in the evaluation and the main results. It also presents and discusses the results obtained in the MatlabEmulator, HelixEmulator, and in the real laboratory testbed.
- *Chapter 7 - Conclusions and Future Work* - this chapter contains the conclusions related to the developed work and also points for possible improvements and future guidelines to continue researching the topic of content distribution in vehicular networks.

Chapter 2

Fundamental Concepts

2.1 Chapter Description

This work covers two fundamental areas of research: Vehicular Ad-hoc NETworks (VANETs) and DTNs. This chapter aims to familiarize the reader with these topics giving an overview of the fundamental definitions and concepts associated with them.

This chapter is organized as follows:

- *section 2.2 - VANETs*: presents the definition and basic concepts of VANETs as well as their architecture, technologies and standards, challenges and applications.
- *section 2.3 - DTNs*: introduces the concept of DTN focusing on its characteristics, architecture, protocols and main applications.
- *section 2.4 - Chapter Considerations*: depicts the conclusions and the summary of the full chapter.

2.2 Vehicular Ad-hoc NETworks

2.2.1 Introduction

2.2.1.1 History

Since the creation in 1991 of the Intelligent Vehicle/Highway System (IVHS) [3], vehicular connectivity and communication started to be envisioned to decrease traffic congestion, enhance road safety and reduce the environment pollution. The IVHS was created in the Intermodal Surface Transportation Efficiency Act (ISTEA) and remained under the responsibility of United States Department of Transportation (USDOT). In 1996 the USDOT, associated with the Intelligent Transportation Society of America (ITSA), structured a framework for the planning, definition and integration of National Intelligent Transportation Systems Architecture (NITSA), which evolved into the current Intelligent Transportation Systems (ITS).

Seeing the relevance of wireless communication for the variety of applications and services proposed and forecast by NITSA, a narrow spectrum band near 900 MHz was allocated. However, the selected spectrum was not appropriated for the majority of applications, whereby in 1999 the Federal Communications Commission (FCC) allocated 75 MHz in the 5.9 GHz

(5.850-5.925 GHz) band to be used exclusively for Dedicated Short Range Communications (DSRC) [4] in North America.

In 2002 ITSA proposed the adoption of a single standard for Physical (PHY) and Medium Access Control (MAC) layers. For this purpose, in 2005 the Institute of Electrical and Electronics Engineers (IEEE) created a development group called IEEE 802.11p [5] task force, which developed standards until 2009. The work of standardization for the remaining layers was developed by another task force that developed the IEEE 1609.x family of standards [6]. Following this standardization activity, in 2008, a decision by the European Commission (EC) established that part of the same spectrum (allocated in North America) should also be allocated in Europe to guarantee the interoperability between those regions. The 5.855-5.905 GHz band was allocated.

In 2010 the IEEE 802.11p standard was finished, giving rise to a set of projects related to VANETs, such as DRIVE-IN [7], Future-Cities [8], SAFESPOT [9], FleetNet [10] and CVIS [11].

2.2.1.2 Definition

In the last years the advances in wireless technologies and the automotive industry lead to the appearance of vehicular networks, also know as VANETs. These networks are spontaneously formed by vehicles (in movement or not) equipped with wireless interfaces which allow the communication between them and the access to other networks.

The nodes of a VANET have to be equipped with an OBU to enable the communication with other vehicles and the infrastructure through an RSU. OBUs enable the dissemination of information by the mobile nodes. On the other hand, RSUs are static nodes deployed along the road or/and strategic locations (e.g. intersections, high buildings). Two types of communication can be established: (i) Vehicle-to-Vehicle (V2V) communication when it is between nearby vehicles and (ii) Vehicle-to-Infrastructure (V2I) when an OBU needs to reach the infrastructure.

A vehicular network can be formed by a heterogeneity of vehicles such as private cars, public services (police cars and ambulances) or public transportation (buses and taxis). The RSUs could be property of the government or private service providers.

2.2.2 Architecture

Vehicular networks can be deployed mainly in three scenarios: urban, rural and highway environments. A VANET could be deployed with at least two vehicles in range of each other which are able to establish a communication between them. Furthermore, a vehicular network could have fixed-infrastructure associated to it used by vehicles to access Internet-based services.

2.2.2.1 Network Components

The architecture of a VANET relies on three main components: OBU, Application Unit (AU) and RSU. According to Al-Sultan *et al.* [12] they could be described as follows.

The **OBU** is the core element of a VANET. These devices are equipped with a Wireless Access in Vehicular Environments (WAVE) communication interface (IEEE 802.11p) and must be installed in each vehicle. Besides that, they could have a wired interface (typically

Controller Area Network (CAN)) to connect to the AU allowing the collection of data. Moreover, they are equipped with a CPU, memory and storage resources, a user interface, and even other wireless communication interfaces such as IEEE 802.11a/b/g or cellular technology. OBUs are mainly responsible for wireless access radio, data security, routing, network congestion control, IP security and reliable message transfer.

The **AU** is equipped within the vehicle and it is used to execute applications using the communication capabilities of an OBU. These applications can include safety and non-safety services. The distinction between an OBU and an AU is mainly logic given that the two components could reside within the same element. The two components can be connected through a wireless or wired interface. Therefore, examples of an AU range from a safety dedicated device to personal computers or smartphones in order to access the Internet always through an OBU.

The **RSU** is usually deployed along the road side or in strategic locations like parking lots or traffic lights. As OBUs, the RSUs are equipped with one or more communication interfaces to communicate with other vehicles (using wireless technology such as IEEE 802.11p or IEEE 802.11a/b/g) and with network infrastructure (using wired technology such as Ethernet or fiber optics).
The main functions performed by RSUs are (i) re-distributing and sending information from or to other RSUs/OBUs; (ii) providing Internet access to OBUs and (iii) running safety applications acting as an information source.

2.2.2.2 Communication domains

Car-to-Car Communication Consortium (C2C-CC) describes in its manifesto [13] the Car-to-Car (C2C) Communication System architecture which is the reference architecture adopted to specify vehicular networks' core components and their interaction. According to this, a VANET architecture can be divided into three domains (as shown in Figure 2.1):

- **In-vehicle domain**: this domain is composed by an OBU and one (or more) AUs. These two components can be connected through a wireless or wired (e.g. CAN) link and, typically, there is one of each per vehicle. The connection between them allows the execution of a set of applications provided by the AU using the communication capabilities of the OBU.

- **Ad-Hoc domain**: this domain consists of a set of interconnected vehicles, each one equipped with an OBU. These vehicles can be also connected with one or multiple RSUs deployed along the road.

- **Infrastructure domain**: an RSU can access the Internet or infrastructural networks enabling the OBU access to the core network. The direct communication from an OBU to the infrastructure is also possible using cellular technology (e.g. Global System for Mobile Communications (GSM), Universal Mobile Telecommunications System (UMTS) or Long Term Evolution (LTE)).

Figure 2.1 also shows the variety of communication possibilities and configurations which are available:

- **Pure Ad-Hoc (V2V)**: there is only inter-vehicle communication (V2V). A node reaches further nodes using other vehicles (multi-hop) with no infrastructure support.

- **Fixed Infrastructure (V2I)**: the new road side equipments (RSUs) could be used to access other networks and farther nodes. Furthermore, the already deployed cellular network and Wireless Fidelity (Wi-Fi) hot spots could also be used.
- **Mobile Infrastructure (V2V/V2I)**: the patterned traffic of public transportation (as buses and taxis) could turn those vehicles into Mobile GateWays (MGWs) to be used as relay nodes or gateways to other networks.

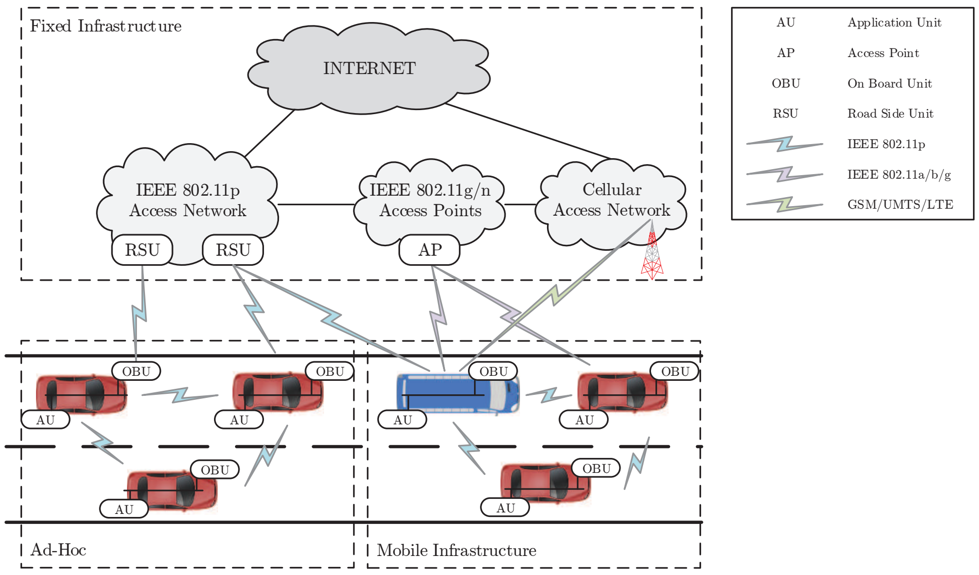


Figure 2.1: VANET architecture, based on [14]

2.2.3 Dedicated Short Range Communications

One of the main questions raised when talking about VANETs is "Why not use the already deployed cellular network and infrastructure instead of DSRCs?". Although the cellular communications are a widespread technology, they are not suitable for the specific environment of VANETs.

The two main advantages of DSRCs against cellular networks are the coverage and cost. The majority of mobile operators do not cover all of the territory, specially the rural and isolated areas where VANETs could easily be used for emergency and safety services. Despite the decline in prices, data plans remain expensive. DSRC communication is free and should be integrated by manufacturers in vehicles allowing inter-vehicle communication anywhere regardless of any mobile operator.

One of the key services for which VANETs were designed was safety services that impose major constraints of latency. However, cellular networks have a higher latency compared to DSRCs as shown in Table 2.1. Although Table 2.1 presents a latency time for Wi-Fi technology close to DSRC, this is not appropriate for inter-vehicle communications due to the complex and slow association process (as will be discussed later in subsubsection 2.2.4.1).

However, the inherent disadvantages of using Wi-Fi and cellular technologies in VANETs do not render their usage impossible. A vehicle could use a Wi-Fi access point when it is moving at a reduced speed or the cellular infrastructure when no other technology is available.

Table 2.1: Comparison of DSRC, Wi-Fi and Cellular technologies [14]

	DSRC	**Wi-Fi**	**Cellular**
Range	100∼1000m	30∼100m	1∼30km
Latency	200μs	3ms	200ms∼3s
Cost	Free or Cheap	Free or Cheap	Expensive

In 1999, the FCC allocated seven 10 MHz channels of spectrum at 5.850-5.925 GHz, for DSRCs. Among these seven channels, one is for control, called Control Channel (CCH), and the other six for service communications, called Service Channel (SCH), as described in Figure 2.2.

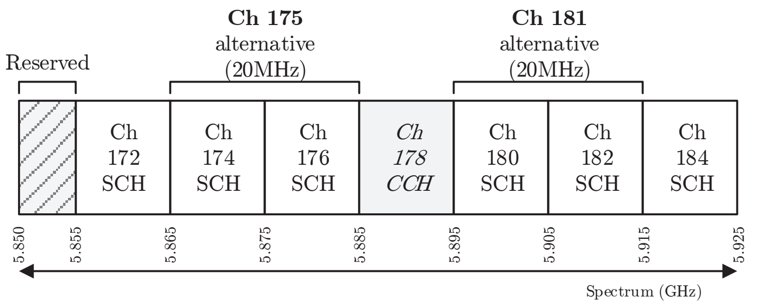


Figure 2.2: Channel allocation for DSRC in United States, based on [15]

2.2.4 IEEE Standards

To regulate the VANETs operation, IEEE developed a new stack of protocols, the WAVE protocol stack, as shown in Figure 2.3. The WAVE protocol stack is mainly composed by these sets of standards: IEEE 802.11p [5] and IEEE 1609.x family [6]. In the following subsection the aim and specific characteristics of IEEE 802.11p and IEEE 1609.x are explained.

2.2.4.1 IEEE 802.11p

An IEEE task force finished the IEEE 802.11p standard in 2010. This standard is an extension of the well-known IEEE 802.11 standard and was created by modifying the IEEE 802.11a standard. According to Jiang and Delgrossi [16], it was developed to be more resilient to the challenges imposed by vehicular networks, especially those related with Quality of Service (QoS), wave propagation and communication outside of a Basic Service Set (BSS) coverage area.

Also according to Jiang and Delgrossi [16], a set of modifications to the IEEE 802.11a standard were performed in the PHY layer. The frequency band was changed from 5.0 GHz to 5.9 GHz and channels of 10 MHz instead of the traditional 20 MHz are used, providing better resilience to multipath and interference. The transmission mask was improved, being more stringent than the required by IEEE 802.11a.

At the MAC layer level the main enhancement introduced by IEEE 802.11p was the simplification of BSS operations making this standard a valid solution for a vehicular environment

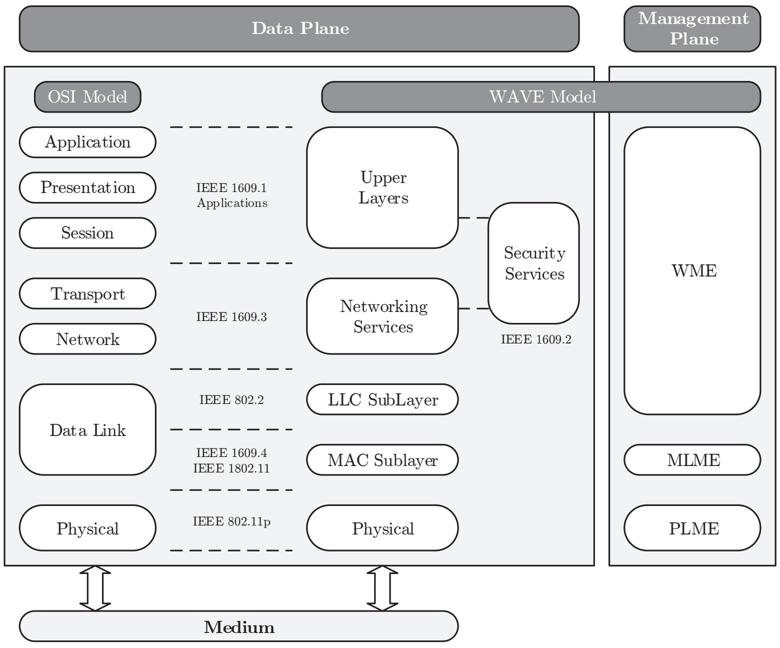


Figure 2.3: WAVE Protocol Suite, based on [14]

working in a truly ad-hoc manner. The introduction of the possibility to communicate outside the context of a BSS is one of the major amendments introduced by this new standard. The previous IEEE 802.11 MAC operations for association/authentication were to time consuming to be suitable for IEEE 802.11p. Figure 2.4 compares the association process using Simultaneous Authentication of Equals (SAE) in traditional Wi-Fi networks with the similar process for initiation of communication in IEEE 802.11p. This standard allows the transfer of limited information between nodes within a specific channel. Among others, a node can receive a service announcement for a certain service and, if it is interested in it, change its configurations according to the received parameters. Other functionalities related with the authentication procedure were passed to the upper MAC layers, defined by IEEE 1609.x.

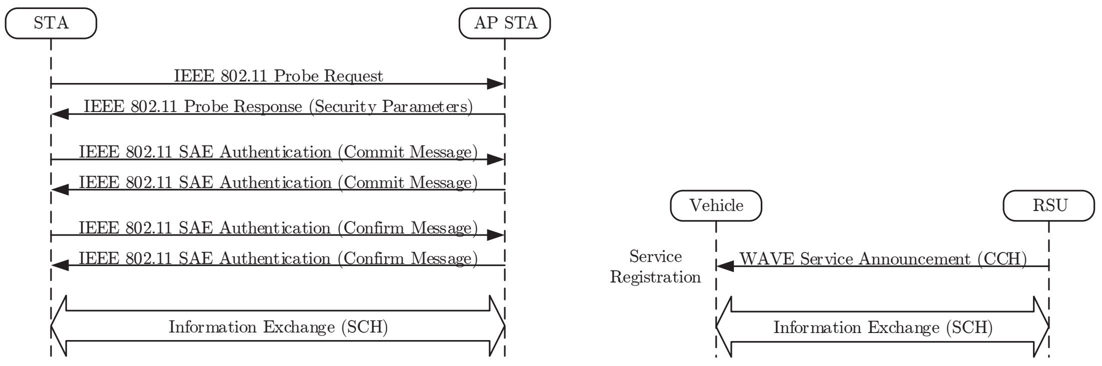
The IEEE 802.11p standard also specifies the management functions associated with the PHY and MAC layers (the Physical Layer Management Entity (PLME) and the MAC Layer Management Entity (MLME) blocks in Figure 2.3).

2.2.4.2 IEEE 1609.x

The IEEE 1609.x family is composed by a set of minor standards which specify the control and management services provided by MAC layer. Each one of them deals with a specific area and is described by Gukholl and Cherkaoui [17] as follows.

- IEEE 1609.1 [18]: describes the key elements of the WAVE system architecture and provides a resource manager for WAVE which specifies the interaction between the network equipments present in a vehicle and remote computing resources to balance the processing levels performed by an OBU.

- IEEE 1609.2 [19]: defines the security mechanisms for services and applications, and


STA
AP STA
IEEE 802.11 Probe Request
IEEE 802.11 Probe Response (Security Parameters)
IEEE 802.11 SAE Authentication (Commit Message)
IEEE 802.11 SAE Authentication (Commit Message)
IEEE 802.11 SAE Authentication (Confirm Message)
IEEE 802.11 SAE Authentication (Confirm Message)
Information Exchange (SCH)
Vehicle
RSU
Service Registration
WAVE Service Announcement (CCH)
Information Exchange (SCH)

(a) Traditional Wi-Fi (b) IEEE 802.11p

Figure 2.4: Comparison of the association processes, based on [14]

specifies the secure messages formats and the circumstances for the exchange of secure messages.

- IEEE 1609.3 [20]: specifies the network and transport layers' services. It describes the Wave Short Messaging Protocol (WSMP), which is an alternative to the Internet Protocol (IP) providing routing and addressing schemes for the service and control channels. The standard also defines a set of management functions implemented in the WAVE Management Entity (WME), which are used to provide networking services.

- IEEE 1609.4 [21]: defines the enhancements to IEEE 802.11 MAC to support a multi-channel operation to allow the usage of service and control channels simultaneously. This standard specifies an access scheme similar to Time Division Multiple Access (TDMA), which allows the switching between the SCH and CCH very quickly. Using this approach it is possible to periodically hear control and data information in CCH and SCH, respectively, independently of the load in SCH. Because of this fast switching, a guard interval of 4 ms was introduced to exchange between channels whereby an accurate synchronization among nodes (could be achieved using Global Positioning System (GPS) technology) is essential.

2.2.5 Special Characteristics

VANETs are a specific type of Mobile Ad-Hoc NETworks (MANETs), as they have their own behavior and unique characteristics. The main difference between them is that in VANETs the mobile nodes are vehicles which, according to Nekovee [22], lead to the following special characteristics:

- **Unlimited battery power**: in a VANET the nodes are vehicles which do not have power supply limitations like in other MANETs such as Wireless Sensor Networks (WSNs). Because of that it is assumed that nodes can provide a continuous and unlimited (theoretical) power to computing and communication devices.

- **Higher computational capability**: due to the unlimited power supply the network nodes can bear a high computational effort related to computing, communication and sensing.

- **Predictable/patterned mobility**: due to the traffic rules (speed limit and traffic lights), underlying roads, drivers' behavior and traffic conditions (rush hours), most of the time vehicles follow a certain mobility pattern. Systems like GPS already provide those kinds of information which can be used to evaluate the current and future position of the vehicle. These data are very useful for a variety of applications, services and network decisions such as routing protocols.

2.2.6 Challenges

The special characteristics inherent to a vehicular environment bring a set of challenges that have to be considered and handled. Those challenges can be divided in two groups: (i) specific to vehicular networks and (ii) associated with technical and technological issues.

2.2.6.1 Specific to vehicular environments

- **Potentially large scale**: in developed countries between 50% and 85% of the citizens have a vehicle, the potential growth of these networks is much higher compared to other typical ad-hoc networks.

- **Highly dynamic topology**: since the network nodes are vehicles that move at a variable speed, the topology formed by VANETs is changing very often. Furthermore, while the vehicles' speed is predictable in motorways and highways, in urban and rural scenarios it is not. In an urban scenario, the large amount of different paths to reach the same destiny and the high number of intersections turns the prediction of a route almost impossible, reducing the effectiveness of network protocols. In addition, the high speed of the vehicles reduce the connection time between nodes, imposing a connection establishment as fast as possible and a higher range of wireless links.

- **Network fragmentation (intermittent connectivity)**: as vehicles are constantly moving the network topology changes very often leading to frequently broken links between the network nodes. Moreover, this problem is exacerbated by heterogeneous node density caused by the variety of scenarios where these networks are deployed. When even the contact with the RSUs failed, the use of the cellular network to mitigate this lack of connectivity (to the core network) is always an alternative; however, it represents an increase in expenditure for the user.

2.2.6.2 Technical

According to Moustafa and Zhang [23], scalability and interoperability are the two main issues to take into account in developing protocols and mechanisms for VANETs. In the following items the main technical challenges to achieve the best scalability and interoperability among the network are identified and discussed.

- **Reliable Communications and MAC Protocols**: vehicles can communicate through other vehicles experiencing multi-hop communication which represent a major challenge on reliability of communication. MAC protocols have to ensure a low communication latency, fast association and have to cope with the high dynamic of the network. Besides that, they must guarantee a fair treatment of safety and non-safety applications.

Furthermore these protocols should take into consideration the heterogeneity of wireless technologies (Wi-Fi and GSM/UMTS) used in a vehicular network.

- **Routing and Dissemination**: regarding the high mobility of the network, the routing schemes for information dissemination must be sufficiently fast and efficient. Besides that, due to the variety of applications and services running in the network, routing algorithms should be adaptable to give different priorities of transmission to different services (safety and non-safety). Furthermore, routing protocols should adapt according to the network's topology, link quality and node density.

- **Security and privacy**: the privacy of the user and the authentication of messages exchanged between vehicles must be protected and ensured from malicious users.

- **IP Configuration and Mobility Management**: the access from vehicles to the Internet through the infrastructure raises these two technical challenges. Due to the characteristics of a vehicular network, the IP configuration should be automatic and non-centralized. Since vehicular devices are equipped with multiple communication interfaces, automatic and optimized connection (and mobility) management is a relevant issue.

2.2.7 Data Dissemination

All applications foresee for VANETs rely on data dissemination among the vehicles with potential interaction of RSUs in the process. There are some important concepts concerning data dissemination that should be clarified.

Data dissemination can be classified according to the number of hops by which the information travels until it reaches its final destination:

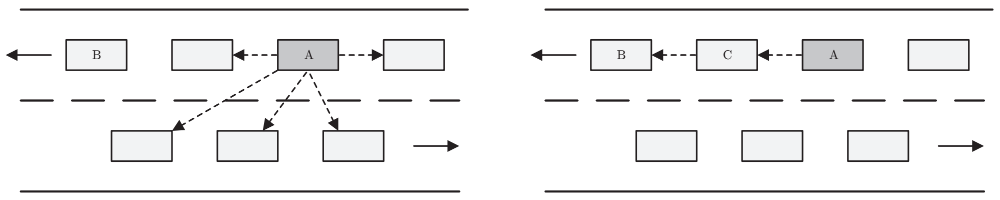
- **Single-hop**: this type of dissemination is implemented with broadcast at MAC layer. As Figure 2.5-(a) shows, a node A can only send information to its neighbors (in the transmission range) which means that node B will not receive the content of the message spread by node A.

- **Multi-hop**: The broadcast of information is closely related to VANETs services and applications, particularly to the dissemination of safety messages which need to be spread through a large set of vehicles using a multi-hop transmission. In these services, data should be transmitted through multiple vehicles until it reaches its final destination. In Figure 2.5-(b) node A sends information to node B, but to reach B, it uses the intermediary node C as a relay node because there is no connectivity between nodes A and B.

Single-hop and Multi-hop are both needed to disseminate information through a vehicular network and could be combined in an hybrid strategy.

According to the final destination of the information, messages can be classified as follows:

- **Unicast**: there is one sender to one receiver of data. The majority of applications related with entertainment (e.g. Internet access, gaming and video-streaming) use unicast messages to disseminate data.



(a) Single-hop (b) Multi-hop

Figure 2.5: Single-hop (a) and Multi-hop (b) data dissemination, based on [23]

- **Multicast**: there is one sender to one or multiple receivers of data. Some safety applications disseminate information using multicast messages when they want to reach a specific direction or region (multicast group).

- **Broadcast**: there is one or multiple senders and data should be delivered to all vehicles. However, according to Kremer [24] the majority of applications only broadcast messages in a delimited region called Zone-of-Relevance (ZoR).

2.3 Delay Tolerant Networks

2.3.1 Introduction

2.3.1.1 Concept

Ma *et al.* [25] defines a DTN as "the area of networking which addresses challenges in disconnected, disrupted networks without an end-to-end connection". These networks were created to ensure communication over extreme environments like space or interplanetary communications. Those environments are characterized by their high latency (which can achieved hours or even days).

The majority of protocols used in the Internet are based on Transmission Control Protocol (TCP)/IP. However, the TCP often does not work in delay and disruption environments due to the fundamental assumptions built into the Internet. According to Cerf *et al.* [2], the Internet was built assuming the following conditions:

- Existence of an end-to-end path between source and destination entities during the communication;

- Small probability of end-to-end packet drop;

- All elements in the network (routers and endpoints) support the TCP/IP protocol;

- Applications do not have any concerns about communication performance.

- The communications reliability (error correction and congestion management) is provided by TCP.

However, in a delay tolerant network, the previous assumptions are flexible whereby the the design principles and requirements are different from the previous.

2.3.1.2 Challenges and Requirements

As mentioned before, there are scenarios in which TCP/IP protocols do not perform well. These environments share some challenging characteristics with current Internet-based networks as the following [26]:

- **Intermittent Connectivity**: in many challenged environments, end-to-end connectivity between source and destination is not always achieved. This issue is called *network partitioning*, and in these cases the TCP/IP protocols do not perform well. So, these new protocols must support communication without a clearly defined end-to-end path.
- **Long or Variable Delay**: the propagation and transmission delay of a link are directly affected by the underlying transmission medium. In some challenging scenarios a long or variable communication latency contributes to an end-to-end path delay that could cripple Internet protocols and applications that required quick transmissions and data acknowledgments.
- **Asymmetric Data Rates**: data rates may be considerable asymmetric between network elements (e.g. high rate for download information but a small uplink for control data). In those cases, conversational protocols used on the Internet will not work correctly.
- **High Error Rates**: the procedures to promote error corrections increase the network traffic due to the retransmission of packets and processing because of the introduction of control data. For the same link-error rate, the number of needed retransmissions is smaller when the communication is hop-by-hop than end-to-end (used on the Internet).

Because of the previous challenges, the protocols to be used in delay tolerant networks face new requirements, and thus need to be built from scratch or adapted from Internet-based protocols.

2.3.1.3 Applications

The DTNs were initially developed to be used for interplanetary communications, although they have a large application field on Earth. Some of these are listed as follows [27]:

- **Terrestrial mobile networks**: in these networks, the nodes' mobility and changes in signal strength might cause the unexpected disruption of the network, or, in other cases, this partition can be periodical or predictable. So, the traditional Internet protocols (such as TCP or IP) do not work well because of the absence of an end-to-end connection. Applications using data mules are typical in this class, and an example is the use of bus to store, carry and forward information among remote villages providing a form of messaging switching service (like mail service). Another possible application is the use of a DTN in VANETs.
- **Sensor/actuator networks**: the network elements are typically sensors (or actuators) with low power, memory and processing capabilities. Moreover, the number of nodes in WSNs is usually high, with possibly thousands or millions of nodes in the network. Due to the power limitations, the communications are often scheduled. Furthermore, they typically employ "proxy" nodes to translate Internet protocols to the usually used

sensor protocols. These properties make this type of networks suitable to employ a DTN approach.

- **Military ad-hoc networks**: these networks operate under extreme conditions where they are often disrupted due to environmental factors, node mobility, and intentional breaks of connection. Moreover, some military applications have to compete for bandwidth with higher priority services. In those cases, traffic applications have to wait for available bandwidth until services with a higher priority stop using the medium. These conditions enhance the use of DTNs.

- **Exotic medium networks**: very long distance optic or radio links, satellite communications, acoustic links in air or water and free-space optical communications. The major common characteristics of these are the high latency, predictable communication disruption, and extreme weather conditions. Due to the set out characteristics, DTNs are a suitable solution to these applications.

Figure 2.6 summarizes the possible applications for a DTN:

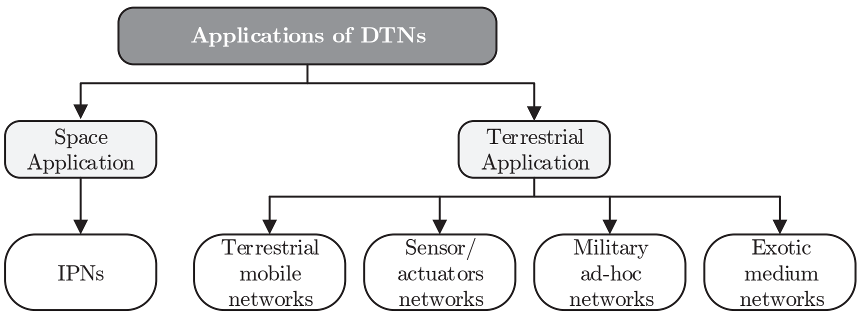


Figure 2.6: DTN applications taxonomy, based on [27]

2.3.2 Architecture

The DTN architecture aims to adapt to network disruption and also to the heterogeneity of underlying protocols used to provide network functionalities such as transport and routing. According to Fall [27], DTN uses layering, naming, encapsulation, and persistence storage to interconnect heterogeneous parts of a network. The following topics discuss the strategies adopted by a delay tolerant network to deal with these issues.

2.3.2.1 Overlay Architecture

To ensure a reliable communication between nodes when frequent network disruptions occur, a new layer called *bundle layer* was implemented above the transport layers of each network node and below the application layers. According to Ma *et al.* [25], the bundle layer forms an overlay that employs persistent storage to deal with network disruptions. This layer also includes a mechanism for reliable communication moving the responsibility of hop-by-hop delivery and implementing an end-to-end acknowledgment procedure. The bundle layer also introduces diagnostic, management, security and interoperability features.

A DTN could use a variety of delivery protocols (TCP/IP, raw Ethernet or serial lines). Each one of these has specific characteristics and uses different semantics in their messages whereby a DTN introduces a new element called Convergence Layer Adapter (CLA). This element provides the necessary functions to carry DTN data units (called *bundles*) on each underlying protocol achieving interoperability.

As Figure 2.7 shows, the DTN architecture relies on a central element called *bundle forwarder* which is responsible for forwarding bundles between applications, CLAs, and storage based on routing decisions. Two types of data units are exchanged between modules: (i) bundles used by storage, CLAs, and applications, and (ii) directives used by routing, management, and applications.

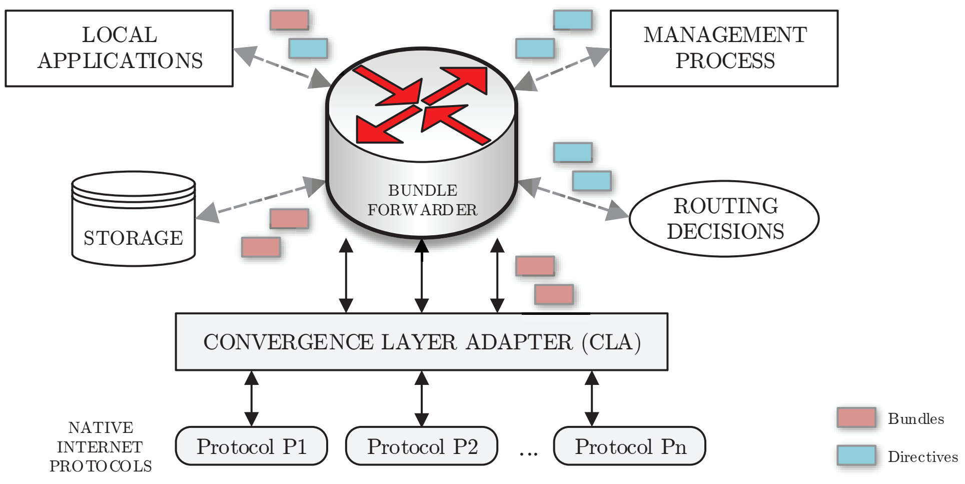


Figure 2.7: Bundle forwarder interaction architecture, based on [27]

2.3.2.2 Store-Carry-and-Forward (SCF) Mechanism

To overcome the issues associated with network disruption (and intermittent connectivity), long or variable delay, asymmetric data rates, and high error rates, DTNs use a mechanism called SCF (illustrated in Figure 2.8). Contrarily to IP networks which are based on "store-forward" mechanism, a DTN needs to store and carry the information while there are no available or reliable links.

To store (and carry) the information, DTN nodes have to be equipped with a storage device (such as a hard disk). Those devices are called persistent storage because they could store the information indefinitely, as opposed to very short-term storage equipments, such as flash memories or routers' buffers. According to [26], the DTN nodes need persistent storage for the following reasons:

- The link which establishes the connection to the next hop could not be available for a long period of time;

- A pair of nodes may have discrepant transmission rates and communication procedures reliableness;

- After sending a message, if it is rejected by the destination node or if an error occurs, the information must be retransmitted.

SCF is the key mechanism used by DTNs to overcome the intermittent connectivity problem.

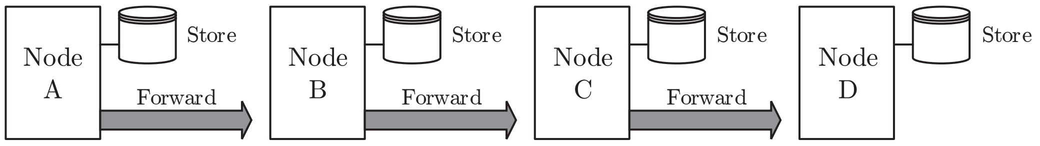


Figure 2.8: Store-Carry-and-Forward mechanism, based on [26]

2.3.3 Bundle Protocol

As mentioned before, to implement a reliable communication among DTN nodes a new layer called bundle layer was introduced. This layer relies on a new protocol standardized by Burleigh and Scott [28], the *Bundle Protocol*. The bundle layer is implemented between the transport and application layers, and allows the communication across the same (or different) set of lower protocols in challenging environments with frequent communication disruption and high delay. All DTNs use the same Bundle Protocol although the lower protocols can vary among the nodes depending on their communication environment.

According to Scott and Burleigh [28], the key capabilities of the Bundle Protocol can be summarized as follows:

- Custody-based retransmission;
- Capability to deal with intermittent connectivity;
- Ability to take advantage of predicted, scheduled, and opportunistic connectivity;
- Late binding of overlay network endpoint.

The previous bundle forward interaction architecture (see Figure 2.7) is implemented by an entity called Bundle Protocol Agent (BPA). This entity is responsible for storing and forwarding bundles between nodes.

The structure of the Bundle Protocol overlay and the comparison between Internet protocol stack and DTN protocol stack is shown in Figure 2.9.

As shown in Figure 2.10, the Bundle Protocol uses the standard protocols of Internet (typically TCP/IP but others can be used) for transporting and routing information under the bundle layer. In a DTN, all nodes have implemented both bundle layer and lower-layer protocols. The forwarding nodes are able to forward bundles between the same or different lower-layer protocols being similar to a Internet router or gateway, respectively.

2.3.3.1 Terminology

There are some terms associated with the Bundle Protocol as found in [28] that should be known:

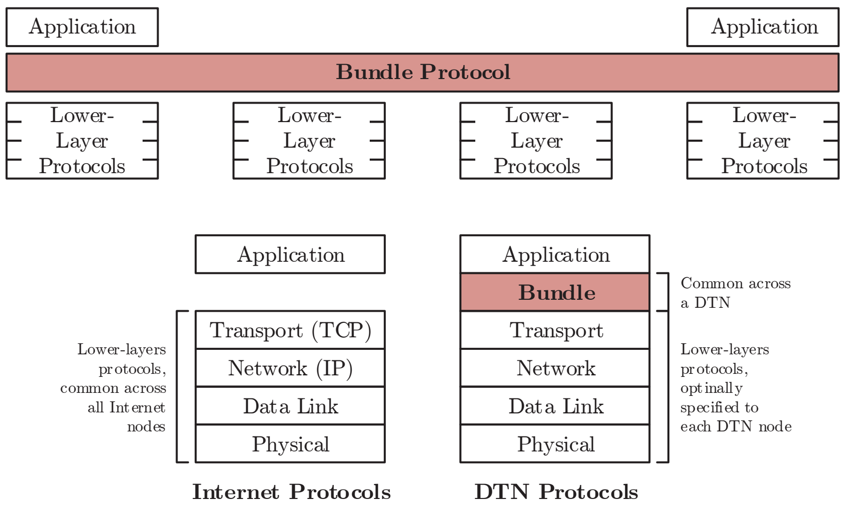


Figure 2.9: Bundle Protocol overlay, based on [26]

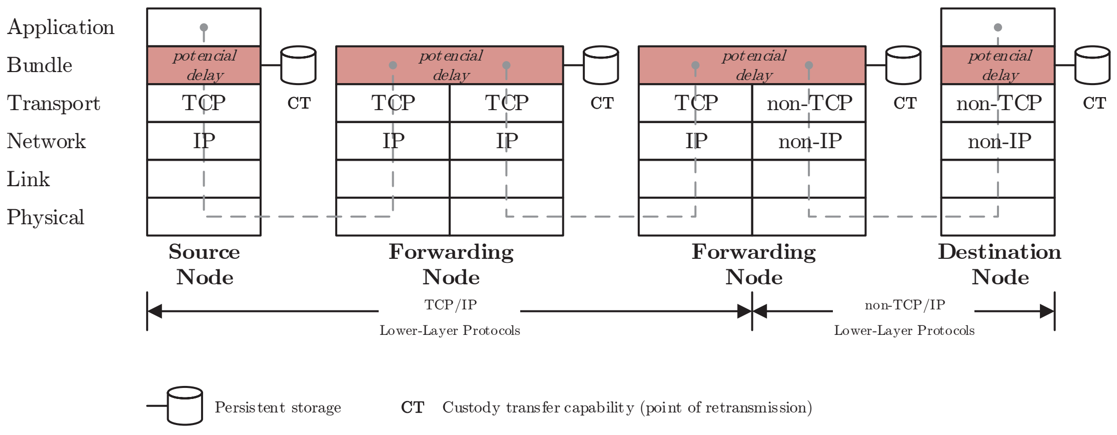


Figure 2.10: Bundle Protocol communication diagram, based on [26]

- **Bundle**: is the protocol data unit of the DTN Bundle Protocol.
- **Bundle Node**: is an entity which can send or receive bundles. A bundle node is composed by a set of conceptual components: a Bundle Protocol agent, one or more convergence layer adapters, and an application agent.
- **BPA**: is the node component which implements the Bundle Protocol providing its services and procedures.
- **CLA**: this layer is used to achieve the convergence between a BPA and lower-layer protocols. It allows the send and reception of bundles by the BPA and also provides the utilization of a native Internet protocol.
- **Application Agent**: is the node component that utilizes the Bundle Protocol services to provide communication and is composed by an administrative element and an application specific element.

- **Bundle Endpoint**: is a set of zero or more bundle nodes which are identified by an Endpoint Identifier (EID).

2.3.3.2 Bundle Service

Six classes of services are provided by the bundle layer:

- **Custody transfer**: delegation of the forwarding responsibility by one node to another, so that the first node recover its retransmission resources.

- **Return receipt**: provides an end-to-end assurance delivery sending a confirmation from the destination node to the source node that the bundle has been received.

- **Custody-transfer notification**: notification sent by the new custodian to the node that previously had the custody of a certain bundle.

- **Bundle-forwarding notification**: notification to the source node when a bundle is forwarded by another node.

- **Priority of delivery**: Bulk, Normal or Expedited.

- **Authentication**: digital signature used to check a node's identity and a message's integrity.

2.3.3.3 Application Data Units, Bundles, Blocks

The messages sent by a DTN application have an arbitrary length and are called Application Data Units (ADUs). The relative order in which a DTN application sends its ADUs might not be preserved until the destination is reached. The bundle layer divides an ADU into protocol data units called *bundles*, which are forwarded by the bundle forwarding entity.

A bundle has a specific format and could contain one or more blocks and each one may contain either application data or metadata to handle network procedures (routing, transport, ...). A bundle could be divided into multiple *bundle fragments* which are also considered bundles by the network and could be re-fragmented. The re-assembling process could be performed by any node of the network at any time.

According to Scott and Burleigh [28], each bundle is composed of at least two blocks which are described as follows.

The **Primary Bundle Block** is the first block in a bundle (and is the equivalent of the IP header on the Internet) and it is used for routing bundles to their destinations.

The **Payload Bundle Block** which contains the payload and information related with it, received from the application layer.

Extension Blocks are all the blocks besides the primary and payload blocks.

All the bundle blocks (with the exception of the primary block) follow a common format with these fields: block type code, block processing control flags, block data length and block-type-specific data fields). The meaning of each one of them can be accessed in [28].

2.3.3.4 Nodes, Endpoints and Registrations

Cerf *et al.* [2] defines a DTN node as an "engine for sending and receiving bundles" which implements the Bundle Protocol [28]. Warthman [26] divided the DTN nodes into two categories based on its function (at that time) in the network (see Figure 2.11).

The nodes with a **source** or **destination** function are the type of nodes which are the source or destination of information and do not forward any bundles. Like any other node in a DTN they need to have persistent storage to store information when they do not have links available to send bundles. Those nodes may optionally support custody transfers.

On the other hand, the nodes with a **forwarding** function act as routers, forwarding bundles between nodes in distinguished situations:

- *Routing-Equivalent Forwarding*: the nodes forward bundles to other nodes which have the same lower protocol stack as the forwarding node (in Figure 2.11 protocols of type "A"). The nodes must have persistent storage and can support custody transfer.
- *Gateway-Equivalent Forwarding*: the nodes forward bundles to other nodes which have a different lower protocol stack as the forwarding node (in Figure 2.11 protocols of types "A" and "B"). The nodes must have persistent storage and can support custody transfer (which in this case is advisable).

DTN nodes are identified by EIDs and each node must have a unique EID. Applications identify the destination by setting the correspondent field in an ADUs with the EID of destination node. Each EID is expressed as a Uniform Resource Identifier (URI).

When an application wants to receive data from a certain node (identified by a unique EID) it is performing a registration procedure.

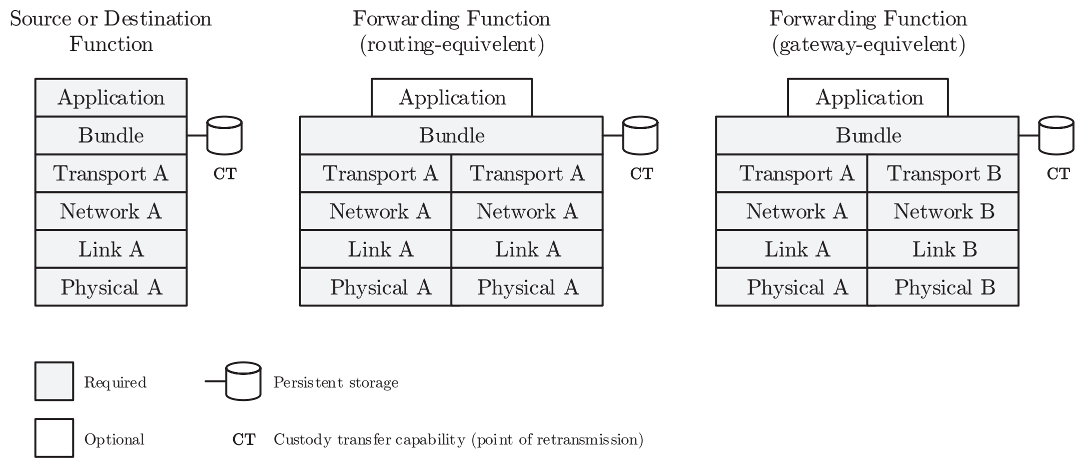


Figure 2.11: DTN nodes, based on [26]

2.4 Chapter Considerations

This introductory chapter described several topics concerning the work already done until now and focused on the main network concepts of this Dissertation: (i) VANETs and DTNs. The most relevant considerations are described as follows.

Vehicular Ad-Hoc NETworks

In this section the concept of VANETs was presented. According to the given description, the positive impact that this type of networks can have in today's society is very relevant leading to a new set of possible applications and services. Regarding this positive impact, many countries are already making efforts on the research and implementation of this kind of networks.

This innovative network architecture is based on mobile nodes (vehicles) which interact with the fixed infrastructure in order to provide a set of applications and services to their users. A vehicular network is characterized by high mobility and typically covers a wide geographical area. Thus, these networks represent an environment with a high potential to provide and support content dissemination services through a significant area of interest, using moving vehicles to carry the data under dissemination to further nodes.

Delay Tolerant Networks

In this section the concept of a DTN was presented as well as its main features and applications. It is clear that the DTN paradigm is gaining relevance and becoming an area of high interest, due to its potential to allow the transfer of information in harsh environments, and its ability to interconnect several network elements which, without mechanisms implemented in DTNs, would not be able to communicate.

The main goal of this work is the development of content distribution strategies to spread data through a vehicular network. Thus, due to the harsh conditions of this type of environment, a DTN will be used in order to assure a reliable dissemination of non-realtime data. This approach leads to an innovation in the manner in which data is spread from a remote content server and sent to the mobile nodes, using vehicles as data mules to transport data.

Chapter 3

Related Work

3.1 Chapter Description

The work developed in this Dissertation covers a considerable set of subjects and research topics such as VDTNs, content distribution schemes, network emulators, or DTN implementations. The main goal of this work is the design and implementation of content distribution strategies in a vehicular environment using delay-tolerant communication to carry and forward data through the vehicles. Once the proposed strategies are designed, it is crucial to evaluate their performance and behavior in a large-scale scenario, whereby it is important to understand which are the available network simulators or emulators to perform this evaluation.

Thus, this work covers several topics from different areas of research. This chapter aims to give a sustained description of these topics along with some related work already developed by the academia and industrial partners.

This chapter is organized as follows:

- *section 3.2 - VDTNs*: presents a set of applications and services that can be offered by a VDTN along with several projects which have already implemented a VDTN.

- *section 3.3 - Content Distribution Schemes*: in this subsection a survey of the most relevant protocols and schemes for content distribution in a vehicular environment is done.

- *section 3.4 - Simulation*: describes a large set of mobility models and network simulators used to emulate and evaluate several routing and network protocols of a vehicular network.

- *section 3.5 - Delay Tolerant Networks Implementations*: describes several widespread DTN implementations along with a solution specifically designed for a vehicular environment by IT and Veniam®.

- *section 3.6 - Chapter Considerations*: depicts the conclusions and the summary of the full chapter.

3.2 Vehicular Delay Tolerant Networks

3.2.1 Evolving from DTNs and VANETs to VDTNs

As discussed before (see sections 2.2.5 and 2.2.6), vehicular environments have special characteristics which introduce several challenges that must be taken into account in the deployment of such networks. The development of protocols for such environments was discussed by Li and Wang [29] where they identified the following major challenges and characteristics of vehicular networks regarding this goal:

- *Vehicular applications*: some safety applications have high delay constraints, for example, the announcement of an approaching emergency vehicle. For those applications, a DTN architecture could not be suitable, precluding the use of a VDTN. On the other hand, the concepts of a DTN can easily be used in applications regarding the dissemination of non-urgent data, for example, transmitting data from sensors to a database located in the *cloud*.

- *High mobility and frequent disconnections*: due to the mobility of the vehicles and their high speed, the network topology is highly dynamic. In addition to that, in low traffic density, sparse networks, or in the presence of physical obstacles (e.g. buildings), the links could frequently be disconnected. Furthermore, the vehicle density can vary from high, in a traffic jam, to low, in a rural area. These characteristics make the use of a DTN architecture very attractive since it was developed to overcome those challenges.

- *Geographical awareness*: in the near future, all the vehicles will be equipped with a location device (e.g. GPS) allowing the determination of its geographical location and the prediction of further trajectories. A typical case is the public transportation which has predictable routes. This data can be used by routing algorithms to better perform its decisions. Furthermore, the VANET's concept of *Geocast* can be used to disseminate information only inside of the ZoR.

- *Storage and computational capabilities*: as the network nodes are vehicles, it can be assumed that they have a unlimited power supply as well as high computational and storage resources.

Thus, the use of a DTN architecture in such networks can be highly beneficial to overcome the previous challenges, whereby the concept of VDTN emerged.

According to Pereira *et al.* [30], the main difference between VANETs and VDTNs is that the former assume an existence of an end-to-end path from source to destination while the latter does not. Accordingly, the concepts of a VDTN are more appropriate for vehicular environments where there is a constant lack of connectivity. Particularly, the use of the SCF mechanism could enable the communication among nodes within these networks.

In a VDTN, the nodes are the vehicles and they are able to store and carry data while there is no available node to forward it. Wherefore, in contrast with other non-VDTNs approaches for VANETs, the vehicles have implemented the Bundle Protocol having the ability of store, carry, and forward information.

With the creation of the VDTN concept some researchers suggested the adoption of two new types of nodes with the goal of increasing the delivery ratio, and decrease the delay on the delivery: *stationary relay nodes* [31] and *data Mobile Uniquitous LAN Extensionss (MULEs)*

[32]. The first type considers fixed nodes mainly deployed in road intersections or isolated regions. These devices are capable of storing information received from mobile nodes and forwarding it to other nodes, or even provide Internet access to the VDTN. In scenarios of low density of vehicles, these new nodes can increase the number of communication opportunities where there is no other alternative (isolated regions) or where there are repeatedly passing vehicles (road intersections). The data MULEs are mobile nodes which transport information along the network. They receive information from a certain region and deliver it in a different location. Using these nodes it is possible to enlarge the network coverage and increase the number of communication opportunities.

3.2.2 Applications and Services

The application field of VDTNs is quite large and includes some of the following examples [30]:

- *Improving road safety* by providing warning and advisory messages to the driver about a possible eminent collision, road conditions or emergency breaks.

- *Optimize traffic flow* informing drivers about the traffic jams and thus preventing road congestions.

- *Sensor networks* for collecting data from sensors and use this information to do statistical analysis, or measure weather and road conditions, parked vehicles in a parking lot, etc.

- *Commercial applications* such as tourist information, marketing data, parking space availability, advertisements, travel, etc.

- *Connectivity to remote areas* such as file transfer, Internet access, e-mail, and telemedicine.

Willke *et al.* [33] conducted a survey of inter-vehicle communication protocols and their applications. They divided the application in four groups as shown in Figure 3.1:

- **Type 1**: this group includes all the applications related to general information services. In this class of applications, delayed or lost information does not compromise user safety whereby DTN concepts and features can improve their quality by increasing the delivery ratio and decreasing latency in challenging vehicular environments;

- **Type 2**: this group is related with services for safety purposes. These services are not delay-tolerant because delayed information can compromises safety. So, these services do not obtain any advantage from the use of a DTN;

- **Type 3-4**: these services require real-time communication and therefore have delay constraints and are also sensitive to the loss of data. As in type 2 services, an implementation of a DTN does not improve this kind of services due to their necessity of permanent connectivity with low delay in communication.

Several examples of these applications are illustrated in Figure 3.1.

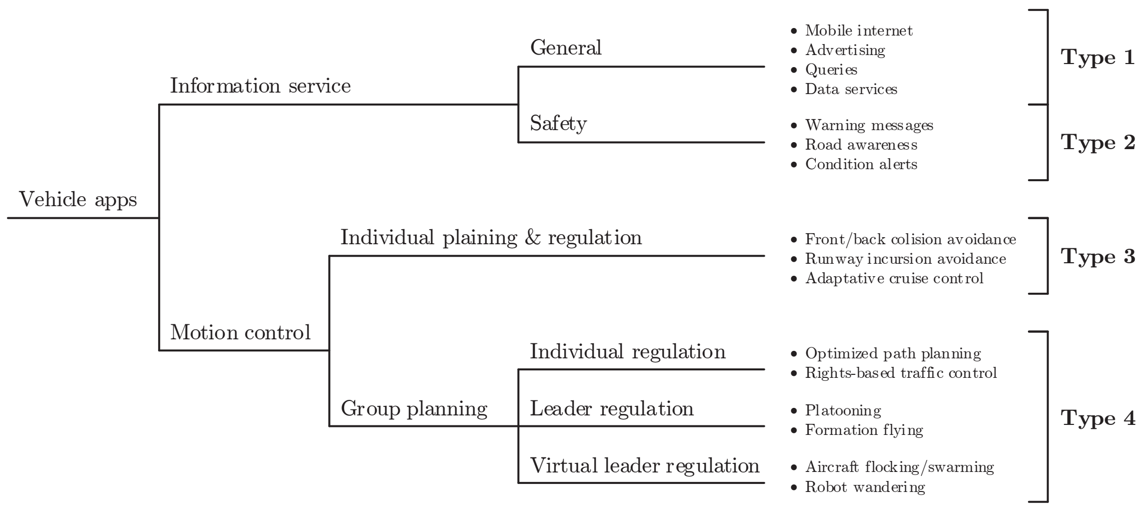


Figure 3.1: Inter-vehicle communication types, based on [33]

3.2.3 Vehicular Delay Tolerant Network Projects

Although the use of a DTN architecture brings the already discussed advantages and improvements, most of the VANET's projects do not use it. Pereira *et al.* [30] conducted a survey on VDTN projects, and in the following subsections their characteristics will be described.

3.2.3.1 KioskNet

The main goal of KioskNet [34] is to deploy a network for Internet access in rural and developing regions. Its network is composed by a set of Kiosks, located in remote villages, and relies on the SCF mechanism to provide Internet-based services to users in those locations.

The authors forecast the use of those Kiosks for a variety of services such as civilian certificates, medical and agriculture services, email, web browsing, and land records. The assumptions for a Kiosk are that it should be easily available, have a reliable Internet connection, and be a low-cost solution. However, due to the intrinsic limitations of these regions (e.g. poor electrical grid and pervasive dust) Kiosk computers often fail. Thus, KioskNet aims to make a Kiosk more robust and still keep the costs low.

The connection to the Internet is assured using a DTN architecture where a bus, which travels around the multiple villages, collects the data from the Kiosks and carries it to an Internet-gateway located in the nearest town. In more detail, a user generates bundles which are stored in a persistent storage until the arrival of the bus. When it arrives, the bundles are transfered to the bus DTN application and they are stored in its storage. Then, when the bus arrives in town, it delivers the bundles trough an Internet-gateway. An overview of the process is illustrated in Figure 3.2.

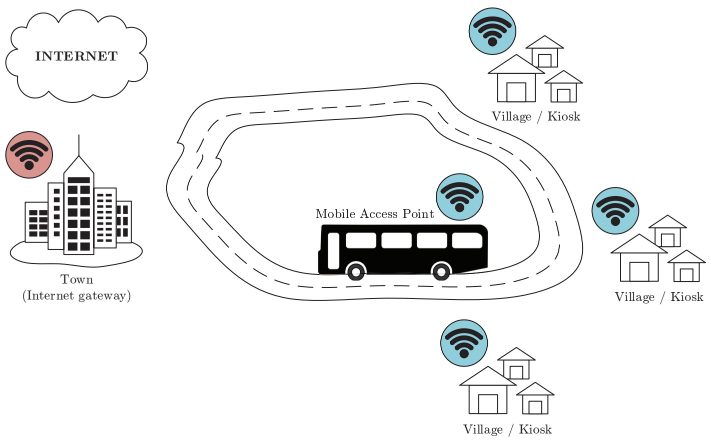


Figure 3.2: KioskNet overview, based on [30,34]

3.2.3.2 DieselNet

The Diverse Outdoor Mobile Environment (DOME) [35] is a large-scale testbed implemented to overcome the problems related with the simulation of mobile networks behavior. Sometimes the simulation processes fail when giving an accurate modeling of the network in terms of mobility, channel and radio characteristics, and power consumption. The DOME system can be divided in three components: the DieselNet vehicular network, a set of nomadic throwboxes, and an outdoor mesh network.

The DieselNet vehicular network provides the mobility to DOME. It is composed by more then 40 public buses which travel across urban and rural environments within an area of 150 square miles. Each bus is equipped with an embedded computer which has multiple interfaces such as a GPS receiver, IEEE 802.11a/b/g wireless interface, and it can operate as an Access Point (AP) trough its IEEE 802.11g interface. Moreover, it also has a wireless 3G USB and 900MHz USB RF modems. The IEEE 802.11 AP allows bus drivers and other buses the establishment of a session, giving them access to the Internet through 3G or radio interfaces. The network of buses is established using the IEEE 802.11 interfaces.

A mesh network was developed to enhance a better connectivity and establish a connection to infrastructure network (fiber optic). This network is composed by 26 Wi-Fi APs which are mounted in strategic locations such as buildings and traffic poles, and supporting seamless hand-offs and are managed by a central controller. The APs mounted in buildings provide direct access to the wired network. On the other hand, the ones that are located in traffic poles guarantee that there are never more than three hops to the wired network.

Due to the size of the testbed and variety of environments where it is deployed, DOME is a rich tool to study issues such as routing, power management, application design and system design for a DTN. Those are the main contributions brought by this testbed.

3.2.3.3 VDTN

Soares *et al.* [36] proposed a layered architecture for VDTNs within the context of the VDTN project. Contrarily to the standard implementation of the Bundle Protocol in a DTN [28], where the bundle layer is implemented above the transport layer of Open Systems Interconnection (OSI) model, in the VDTN project it is placed below the network layer. The authors proposed an IP-over-VDTN layered architecture (see Figure 3.3) to enable the routing of large sized message instead of small sized IP packets which results in a lower complexity, energy savings and lower cost, since there are less packets to route and process.

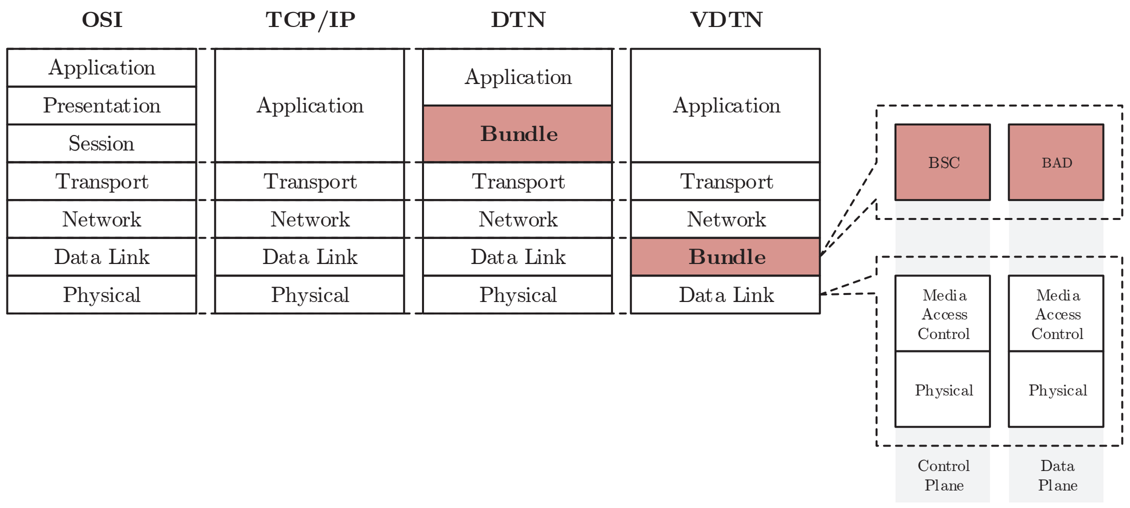


Figure 3.3: IP-over-VDTN layered architecture (comparison with other protocol stacks), based on [30,37]

As shown in Figure 3.3, the proposed architecture establishes two distinguished planes for control and data. The logic separation is achieved using a process similar to the Optical Burst Switching (OBS) [38]. The Bundle Aggregation and De-aggregation (BAD) layer is responsible for the implementation of the SCF mechanism of Bundle Protocol. When a node wants to send information, the BAD layer aggregates the incoming IP packets into bundles to be transfered to the data plane. In the destination node, the BAD is responsible for the inverse process, de-aggregating the received bundles. In the control plane, the Bundle Signaling Control (BSC) provides a signaling protocol to be used at the connection setup phase. This protocols enables the exchange of intrinsic informations of nodes (e.g. node type, speed, storage capacity, etc.) among each other. Nodes use this data to set an agreement for data transfer that will occur in the data plane, perform routing decisions, set security requirements, and specify traffic prioritization, among others.

The VDTN project establishes the existence of three different nodes: *terminal nodes*, *mobile nodes* and *relay nodes*. Terminal nodes can be located in remote locations and act as APs to the VDTN, proving non-real time application to end-users. Mobiles nodes (vehicles) transport data between terminal nodes. Relay nodes are fixed devices located in strategic places to allow the passing mobile nodes to collect and retrieve information.

The main contribution of the VDTN project was the separation of control and data planes showing that this approach optimizes the usage of resources (e.g. storage and bandwidth) and save energy. In addition to that, in this project the use of stationary relay nodes, location of nodes, geographic routing schedule dropping policies, and caching mechanisms were study in order to achieve a greater communication efficiency.

3.2.3.4 Comparison among multiple projects

Pereira *et al.* [30] summarizes the main characteristics of the previous (and others) VDTNs projects in Table 3.1.

Table 3.1: VDTN project's characteristics [30]

Project	Applications	Protocol Stack	Routing Protocol
KioskNet [34]	Internet access for rural sites	DTN standard stack	Epidemic [39]
DieselNet [35]	Internet access for buses	DTN standard stack	MaxProp [40], RAPID [41], or others
VDTN [36]	Internet access for vehicles	Bundle layer below network layer. Separate data and control planes	Epidemic [39], Spray-and-wait [42], or others
CarTel [43]	Detection of road pavement defects	Mule adaption layer below network layer	Static, Epidemic [39]
EMMA [44]	Pollution measurements, Traffic information	DTN standard stack	Epidemic [39]
Drive-thru Internet [45]	E-mail, Web browsing	Session layer above the transport layer	Through infrastructure
CONDOR [46]	E-mail, Web browsing, IRC, Voice Mail	DTN standard stack	Static
Future-Cities [8]	Sensory data collection	DTN standard stack	Static

3.2.4 Summary

VANETs face some problems due to the high mobility of their nodes, presence of buildings and physical obstacles, which leads to an intermittent connectivity and network disruption. The DTN concept aims to provide an end-to-end connectivity between network nodes in harsh environments such as the vehicular one. Thus, a vehicular network which applies the SCF mechanism of DTNs to store, carry and forward data, is called a VDTN. The introduction of this concept expands the portfolio of applications and services of a vehicular network, introducing new services such as content distribution of non-urgent information (e.g. videos, adds, touristic information, etc.).

In this section several VDTN projects are presented and described. As discussed, the number of VANET projects that already use the concept of DTN is not large, whereby there is a need for new approaches and implementations. Thus, taking advantage of the vehicular testbed in Oporto, composed by more than 600 vehicles, and using the Helix software specifically created by IT and Veniam® to provide delay-tolerant communication capability for those vehicles, a new initiative can be addressed. The main goal is to design, implement, and deploy a set of new services and capabilities for the transfer of non-urgent data.

This work aims to introduce a new type of service to this network allowing it to distribute a non-urgent content (e.g. video, adds, etc.) located in a remote server through the network, using vehicles to carry and forward this information.

3.3 Content Distribution

3.3.1 Introduction

The main drivers for the continuous development of VANETs were the safety applications and services. However, in the past years, with the consolidation of several VANETs projects (such as the DRIVE-IN and Future-Cities), new interests have emerged regarding the introduction of new applications and services. The non-urgent content (e.g. videos, ads, touristic information) dissemination can be included in this new set of services.

The concept of content distribution is an older trend and a well-established service in the wired networks. In these types of networks several client-server schemes have been proposed and successfully implemented. It would be normal to think on applying the same schemes in a vehicular environment. Regarding this question, let's take a look at the following example. An Internet server has a specific video available for download through an AP deployed on the road-side. Thus, when a vehicle (with communication capabilities and running the same client-server protocol) stops in front of this AP, it can easily download the content. However, if the vehicle passes through the AP without stopping, the majority of these client-server protocols will not work due to the insufficient contact time. Regarding this, it is possible to conclude that a feasible solution for a vehicular environment is a Peer-to-Peer (P2P) file sharing mechanism, where multiple vehicles can download parts of the content through the VANET fixed-infrastructure and exchange those pieces to other nodes in order to complete the download [47].

As stated in Chapter 2, VANETs are a specific case of MANETs with the particular characteristics discussed in section 2.2. A pure MANET in a vehicular environment occurs when the vehicles are isolated from the fixed-infrastructure, being grouped into clusters. Thus, the majority of the studies performed on this kind of network can also be applied to the vehicular environments, whereby some of the related work present in this sub-section refers to MANETs.

This section aims to give an overview of several content distribution schemes for a vehicular network which can be deployed in a P2P system, using network coding to enhance their performance, or even other strategies to achieve specific requirements as QoS. There is also a set of key factors identified that must be addressed in a content distribution scheme design.

3.3.2 Content Distribution Schemes

3.3.2.1 Peer-to-Peer Schemes

Before describing several content dissemination schemes which are used in a P2P scheme, it is important to define what is a P2P-MANET. According to Gerla and Lindemann [48], a MANET is formed by several mobile nodes which self-organize themselves without using any pre-existing infrastructure. Moreover, in a MANET, the deployed applications are not *client-server*, but instead display a P2P profile. Due to the previous reasons, the networking performed in a MANET is often called P2P networking [48].

The most commonly used protocol of P2P file swarming is BitTorrent [49], which splits a file into smaller pieces. A peer discovers which pieces its neighbors have and which pieces they are missing, and promptly exchanges them in order to complete the download. However, BitTorrent does not work directly on wireless networks since it was designed to be used in the Internet where link stability is not a major issue. According to Klemm *et al.* [50], the maintenance of a static overlay connection constitutes the major drawback in the deployment of a typical P2P file sharing architecture such as the BitTorrent. Thus, instead of using static overlays, the content chunks should be exchanged between the physical neighbors. Following this approach, several P2P schemes have been proposed, and some of them are described as follows.

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

One of those schemes is proposed by Das *et al.* and is called SPAWN [51,52]. This simple cooperative strategy for content delivery uses a proximity criteria (instead of the conventional rarity metric) for piece selection. In [51], it was shown that SPAWN performs better than the "rarest first" criteria of Internet schemes. Figure 3.4 illustrates an example of the evolution of a content in a vehicle using the SPAWN strategy. Thus, (1) a vehicle is in the range of the fixed infrastructure, and (2,3) starts downloading the content, which ends when the vehicle is out of range (4). After this, (5) it starts gossiping with its vicinity about its own storage content and (6) exchanging chunks of the file with its neighbors, getting a larger portion of the content.

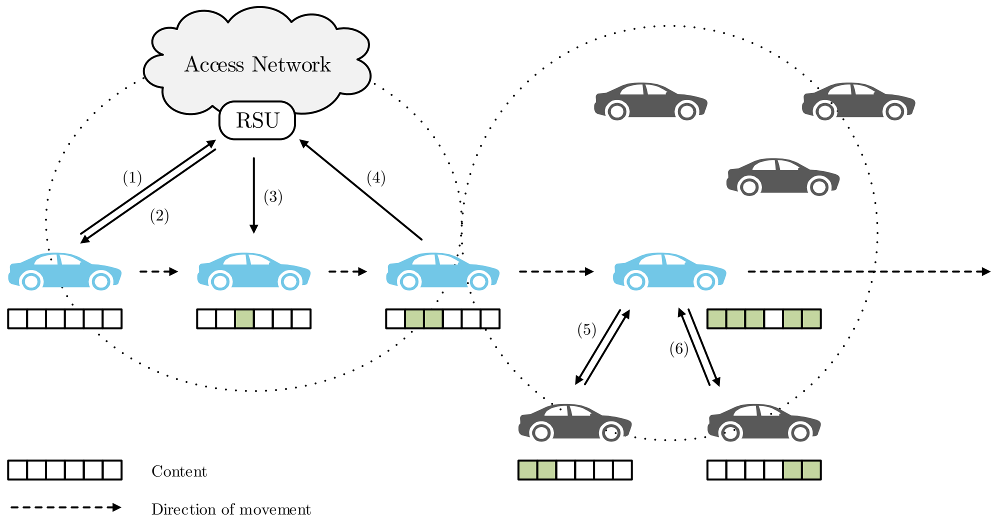


Figure 3.4: Evolution of a file in a node using SPAWN strategy, based on [51]

CarTorrent

Lee and Yap [53] proposed the CarTorrent strategy which can be considered as an extension of the SPAWN. CarTorrent is a file swarming protocol based on BitTorrent deployed for a vehicular environment. CarTorrent is a Java application which uses Ad hoc On-demand Distance Vector (AODV) [54] to perform the route discovery and maintenance along with sending and receiving gossip messages and regular files. This protocol propagates gossip packets up to k hops from the source. These packets are used to allow peers to collect information about piece availability and network topology. This information is used to select the piece that a node will request next under a *rarest-closest-first* policy. In this policy, each vehicle first evaluates the rarest file piece it needs and then looks for the closest node that has it.

REceiver-based solution with video transmission DECoupled (REDEC)

Rezende *et al.* [55] proposed the REDEC, a receiver-based solution that conducts video transmissions decoupled from the relay nodes' selection mechanism. This scheme was designed for the specific case of video streaming. However, its approach to select the relay nodes is relevant to analyze. This strategy aims to combine the reactiveness of a receiver-based solution and an improvement on the relay-node selection mechanism.

In REDEC, a node broadcasts a packet to its vicinity and, when the receiver nodes collect this packet they decide if they are responsible or not for further broadcasting. In order to minimize the size of the packets, REDEC decouples the selection of relay nodes from the transmission of data packets. As illustrated in Figure 3.5, the network can follow one of four states: scheduled, schedule relay, non-relay, and relay. Only the relay nodes are authorized to broadcast the packet, and the non-relay nodes are not authorized but they are interested in the content under dissemination. On the other hand, a schedule node or a schedule relay node are the ones which recently received a control packet and are waiting to become a relay node.

Only the relay nodes are able to broadcast a control packet, whereby, when a node listens a certain amount of control packets it represents the proximity of a relay-node, and there is no need for another one. This refined selection of relay nodes tends to minimize the network congestion and the number of transmissions.

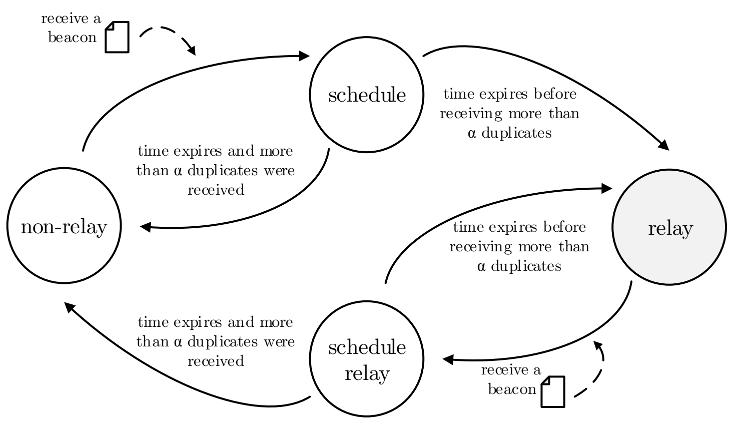

receive a beacon
schedule
time expires before receiving more than α duplicates
time expires and more than α duplicates were received
non-relay
relay
time expires before receiving more than α duplicates
time expires and more than α duplicates were received
schedule relay
receive a beacon

Figure 3.5: Diagram of node's state in REDEC strategy, based on [55]

Mobility-Centric Data Dissemination (MDDV)

Wu *et al.* [56] proposed an algorithm for data dissemination in a vehicular network that combines pure opportunistic forwarding the trajectory-based and geographical forwarding. This strategy is called MDDV. Even though it was developed for pure V2V communication, the authors claim that additional V2I communication can be used for improving performance and functionality. In MDDV the packets are forwarded through a predefined trajectory, and the intermediate vehicles must buffer and forward the packets opportunistically, whereby the

MDDV imposes the nodes who can transmit and when this operation must be performed (or not).

The MDDV aims to improve the efficiency of the delivery through the introduction of traffic flow theory, spreading of messages about the dissemination status, and applying data propagation analysis. The authors also proposed a refined use of the traffic flow theory in order to improve current performance.

Roadcast

Zhang *et al.* [57] proposed a P2P content sharing scheme (called Roadcast) for VANETs. This scheme aims to fulfill two main goals: (i) matching vehicles' query, and (ii) increasing data accessibility in the future. In order to achieve these goals, the Roadcast protocol considers the popularity factor of the data and reflects this in the queries, ensuring that the most popular data is more likely to be shared with other nodes. Moreover, this scheme promotes the storage of data replicas which can be shared among other nodes. Since the storage capacity of the nodes is limited, a set of replacement algorithms are implemented, ensuring that more popular content is present in a higher number of nodes.

PYRAMID

Yu and Bai [58] proposed the PYRAMID, which according to them is a "probabilistic abstraction of contents contained in the vehicles". They resort to a suite of probabilistic data structures to approximate content with different granularities. This implementation aims to answer the following questions: (i) which neighbor has the biggest contribution, and (ii) which packet of a specific content should be transferred. In order to overcome those challenges, two mechanisms were developed: (i) *task prioritization*, and (ii) *content reconciliation*. The first one helps to identify the appropriate transaction partners among a set of vehicles. The other one uses a membership test to avoid transmitting redundant contents. In order to implement these mechanisms, a probabilistic representation of the nodes' storage content must be exchanged among vehicles. The authors confirmed that with the PYRAMID abstraction, the fact that two vehicles are in range of communication and exchange coarse-granularity sketches and fine-granularity summaries of the storage content before starting the exchange of data packets, leads to an improvement of the efficiency of vehicular P2P systems. Moreover, this approach has the cost of additional communication and small computational overhead.

Others

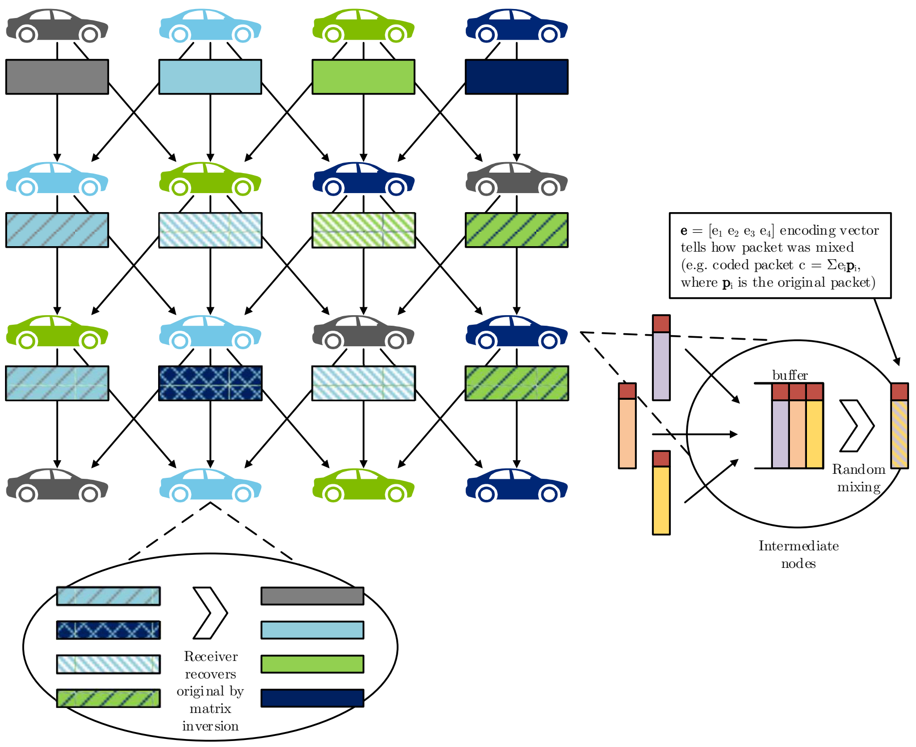
Several other works have been proposed to handle the content distribution challenge using a P2P (pure or hybrid).

Frenkiel *et al.* [59] proposes the Infostation which is used to provide low-cost short-range communications to passing vehicles. From an Infostation vehicles can download voice messages, faxes or e-mails. Yuen *et al.* [60] investigated the use of this concept for non-cooperative content sharing, allowing vehicles to download a file from the Infostation and exchange it among nodes in an opportunistic manner.

Nandan *et al.* [61] proposed the AdTorrent which is a content distribution application to disseminate advertising information in a limited local area. In this scheme, digital billboards with wireless capabilities are deployed along the roadside and are continuously showing advertising contents (e.g. hotel virtual tours, movie trailers, etc.). This content is also disseminated to the passing vehicles which then exchange it in a P2P scheme using a pure V2V communication.

3.3.2.2 Network Coding Schemes

The main goal of a content distribution scheme is to distribute the content as fast and as reliable as possible. According to Gkantsidis and Rodriguez [62], the issue of an efficient content distribution scheme can be modeled using graph-theory, the network nodes being the vertices and the edges of the graph the connections. Thus, a key question comes up: can the optimal throughput be achieved in the process of content delivery from the server to each one of the receivers? Ahlswede *et al.* [63] answers that question, showing that the theory of network coding allows optimal throughput along with computational efficiency. When network coding is applied, the nodes create and forward encoded information along the edge of the graphs to their neighbors. This procedure is illustrated in Figure 3.6.



$\mathbf{e} = [e_1\ e_2\ e_3\ e_4]$ encoding vector tells how packet was mixed (e.g. coded packet $c = \Sigma e_i \mathbf{p}_i$, where $\mathbf{p}_i$ is the original packet)
buffer
Random mixing
Intermediate nodes
Receiver recovers original by matrix inversion

Figure 3.6: Network Coding, based on [64]

In 2005, Gkantsidis and Rodriguez [62] proposed the use of this concept for large scale content sharing in the wired Internet using a P2P system, showing that network coding improves both speed of content distribution and system reliability. Following these studies, the use of network coding for content distribution in VANETs is a recent field of study which has presented very satisfying results. Several schemes and strategies which use network coding for content dissemination are presented below.

VANETCODE

Ahmed and Kanhere [65] proposed a content distribution scheme specifically designed for a vehicular environment called VANETCODE, which relies on the network coding concept [62,63]. The authors propose a division of the content under dissemination into smaller blocks which are stored in road-side gateways which act as content servers. Each server produces a linear combination of the blocks using randomly selected coefficients. The result of this operation is then shared through the nodes' vicinity. The proposed scheme uses broadcast communication to distribute the encoded blocks amongst one-hop neighbors. According to the authors, by using this scheme there is no need to perform a peer or piece selection, therefore optimizing resource usage.

Figure 3.7-(a) illustrates a scenario where this strategy is applied. There are four vehicles (A, B, C, and D) within the communication range of the gateway, and all of them are requesting the content. The content is divided in two blocks (B_1 and B_2), and each one of them in also divided in smaller pieces B_{11}, B_{12}, B_{21}, and B_{22}. This division reduces the number of coefficients required for encoding. After the request, the gateway randomly encoded the pieces by combining the first elements of each block (B_{11} and B_{21}) with an encoded coefficient, C_{11} and C_{12}, respectively. Similarly, it also encoded the second pieces of each block. After that, the gateway sends the encoded block (together with the encoded vector) to the requesting node. This procedure is repeated for all of the requesting vehicles by using different encoded coefficients (C_{21}, C_{22}, C_{31}, C_{32}, C_{41}, and C_{42}).

Another scenario of application is described in Figure 3.7-(b), where all vehicles are outside of the communication range of the gateway. In this case, the vehicles do not wait for the next gateway and share their data blocks amongst each other. Contrarily to the P2P content distribution which uses mechanisms for piece selection, in this scheme there is no need to explicitly request a specific block since all of them are linearly independent, whereby any block received from a neighbor is beneficial. This sharing procedure is similar to the one performed by the gateway in Figure 3.7-(a) since node C peeks random coefficients and linearly combines all of the blocks currently in its storage.

The decoding process occurs when a node captures a sufficient number of blocks with linearly independent coefficients, since this decoding is performed by solving a set of linear equations.

This scheme was one of the first works that showed a practical implementation of the network coding concept for content dissemination in a vehicular environment. Because of this fact, a detailed description of it was performed.

CodeTorrent

Also using the network coding concept, Lee *et al.* proposed CodeTorrent [66], which can be considered as an extension of the work developed by Gkantsidis and Rodriguez [62] but applied to a vehicular network. The authors designed an entirely new protocol aiming to solve the majority of the problems caused by the use of MANETs-P2P protocols in a VANET. The design of this strategy is based on random linear network coding and mobility assisted data propagation as presented by Bai and Helmy in [67].

The mechanism of content distribution can be summarized as follows. A sender node, which has a content to disseminate, initially broadcasts the content description (number of pieces, file identification, etc.) to its vicinity. The sender node also divides the file into smaller pieces ($p_1 ... p_n$) which are coded in groups (using a randomly generated encoded vector $e = [e_1 ... e_n]$), giving origin to coded frames (c) ($c = \sum_{k=1}^{n} e_k p_k$). The nodes exchange coded frames instead of file pieces. Every time a node sends a new coded frame, the encoded vector e is randomly selected from a finite field which is attached to the coded frame. This last behavior justifies the name of random linear coding. In order to recover n file pieces, a receiver node must collect more than n coded frames which must have encoded vectors e linearly independent of each other. This strategy also deployed a recoding procedure in the

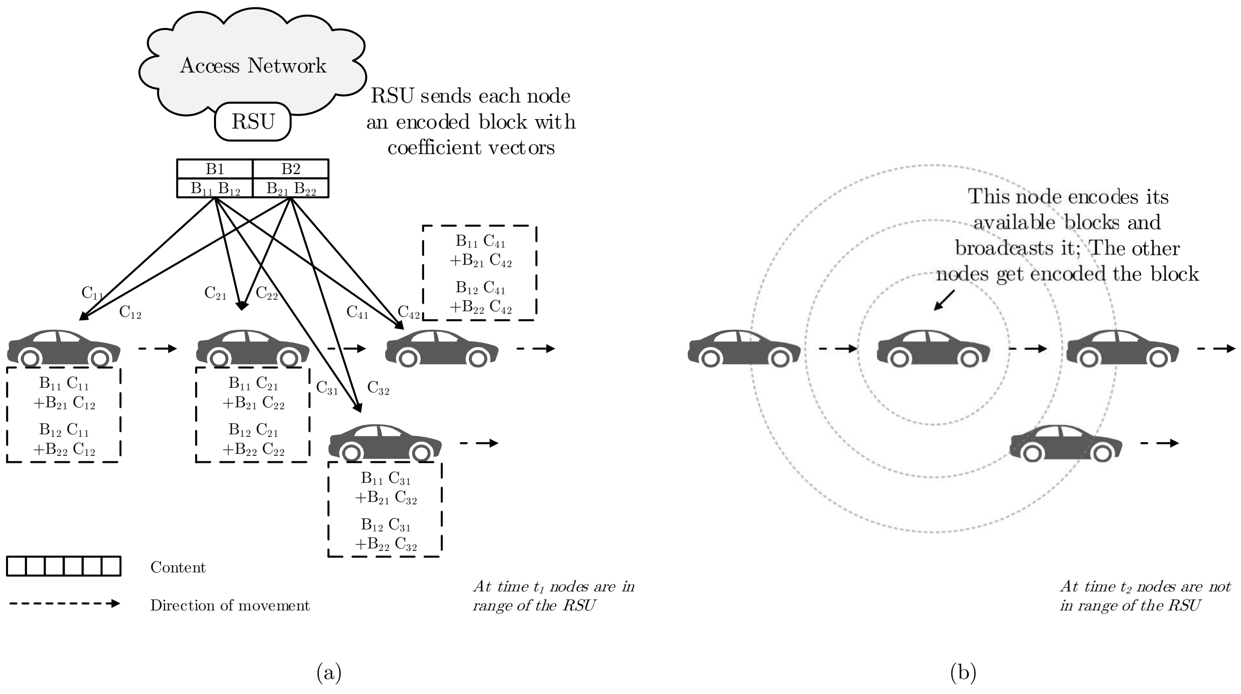

Access Network
RSU
RSU sends each node an encoded block with coefficient vectors
B1 B2
B11 B12 B21 B22
C11 C12 C21 C22 C41 C42 C31 C32
B11 C11 +B21 C12
B12 C11 +B22 C12
B11 C21 +B21 C22
B12 C21 +B22 C22
B11 C41 +B21 C42
B12 C41 +B22 C42
B11 C31 +B21 C32
B12 C31 +B22 C32
Content
Direction of movement
At time t1 nodes are in range of the RSU
(a)
This node encodes its available blocks and broadcasts it; The other nodes get encoded the block
At time t2 nodes are not in range of the RSU
(b)

Figure 3.7: Encoding and distribution of contents in VANETCODE, based on [65]

intermediate nodes, which is identical to the one already described for the initial sender node.

The authors compared the proposed scheme with the CarTorrent, achieving a lower download delay along with a more robust behavior in scenarios that are characterized by a high mobility. However, they concluded that their approach does not completely solve the issues in the VANET P2P system, but it is an easy way to overcome some of those challenges.

CodeCast

Some of the same authors that designed CodeTorrent also proposed a content distribution scheme using the network coding concept, called CodeCast [68]. According to the authors, this protocol is specially tailored for applications with low loss and latency constraints such as video/audio streaming. Similarly to the CodeTorrent, CodeCast also uses random network coding to achieve loss recovery and path diversity keeping the network overhead at low values.

The packets' coding, recoding and decoding procedures are equal to the ones implemented in the CodeTorrent, whereby these will not be described here. Regarding this process, the authors made some considerations about the *blocksize* and *gensize*, which are defined as the size (in pieces) of each file's block and number of coded packets per block, respectively (in [68] *blocksize* is equal to *gensize*). The authors claim that the bigger the *blocksize*, the greater both the efficiency gain and the delay (since a node needs *blocksize* coded packets to decode a block).

This work introduces a new concept called ranking. When an intermediate node does not have *blocksize* pieces to generate a new coded packet, this node combines a lower number of pieces to yield a coded packet and an additional field is recorded as the *rank* in the header of the coded packet. Thus, a coded packet with a *rank* smaller than *blocksize* indicates that the sender node is in need of more coded packets of this specific block. Once this information is gathered, a node can send coded packets of this generation in order to solve the detected problem.

3.3.2.3 Multi-technology Schemes

Several works of content dissemination in vehicular environments use additional communication technologies and are not focused on the IEEE 802.11p and WAVE standards discussed in Chapter 2. This subsection aims to highlight some of the characteristics related with those works, and make some considerations in order to understand their feasibility in non-urgent content dissemination.

In the previously presented works, the Wi-Fi technology is recurrently presented as a possible communication technology. However, it also introduces a set of challenges as listed in subsection 2.2.4. Thus, the option should mostly be the use of DSRC technologies which bring a set of advantages and enhancements when compared to the Wi-Fi technology.

The other set of technologies that are mostly used for content dissemination in vehicular environments are the cellular technologies. Due to its high availability and considerably high throughputs (specially with LTE), these networks have been studied as a mean to disseminate information through a vehicular network. However, the use of cellular networks brings additional challenges and drawbacks that must be considered in the design of content dissemination schemes. Gerla *et al.* [47] identified three main challenges. The first is the fact that cellular networks use point-to-point dedicated channels which can be noisy and lossy leading to problems related with the TCP window. The second one is the fact that the spectrum associated with those networks is limited and its expansion in very expensive. Finally, the most relevant drawback is the high connection cost when compared to the WAVE technology or even with Wi-Fi, which will lead to free-rider problems similar to the ones of BitTorrent [69].

Gerla *et al.* [47] proposed the use of CarTorrent and CodeTorrent considering the availability of both Wi-Fi and LTE radio interfaces. Their work is focused on real time multimedia content download, whereby they state that the use of LTE (cellular network) brings advantages when compared to a pure Wi-Fi approach. Moreover, they concluded that the ideal deployment is a hybrid scheme which uses Wi-Fi for V2V and V2I communications along with LTE only for V2I communication. The authors concluded that a hybrid approach should be deployed since the reliance on only one technology is not effective. For example, the simultaneous download by a large number of vehicles from the same cellular tower could lead to congestion and blocking. On the other hand, the Wi-Fi solution fails in sparse vehicular networks and scarce APs availability. Thus, a synergy between the two technologies is highly recommended. To overcome the challenges associated with the cellular network, the authors proposed the use of network coding to strengthen the LTE connection, the selection of main peers (which first download pieces directly from the cellular repeater and afterwards disseminate it using a P2P system). The authors also suggested the use of reputation and locked out the free loaders to overcome the free riding problem.

Other works have been done using multiple technologies for content dissemination, especially for multimedia and emergency applications. Atat *et al.* proposed a scheme for delay-sensitive content distribution via P2P collaboration using LTE for long-range communications and Wi-Fi for short-range communications. Also, for the distribution of delay-sensitive content, Lee *et al.* proposed Cooperative Video Streaming over Vehicular Networks (CVS-VN). This scheme uses the cellular network (3G/3.5G) and DSRC to enhance a better QoS during a video-stream.

Although the use of different technologies, especially the cellular ones such as UMTS or LTE, bring additional advantages and enhance delivery throughputs, their use is not relevant in the context of this work. The main goal of this Dissertation is the implementation of several strategies for the dissemination of non-urgent content. Thus, the high cost associated with those technologies does not justify their use for non-urgent content download. However, as a future insight it is important to understand that these technologies can support the introduction of new services and applications.

3.3.3 Critical Factors for Dissemination

Regarding the previous stated works and additional survey, it was clear that there is a set of critical factors which directly impact a content dissemination scheme in a vehicular network using a hybrid P2P strategy. It is considered a hybrid P2P system, since in the vehicular environment the content dissemination scheme resorts to the fixed-infrastructure using V2I communication along with a V2V communication through a pure P2P system. The following main factors are identified: (i) network mobility and density, and (ii) piece and peer selection.
.

3.3.3.1 Network Mobility and Density

VANETs are a specific set of networks inside of MANETs. According to Bai and Helmy [67], mobility is a major factor that impacts the performance of MANETs, whereby it is possible to conclude that this impact is also relevant in the specific case of VANETs, since their high mobility is one of their main characteristics. Bai and Helmy also claim that, although the mobility adds several challenges, it also provides opportunities to enhance the performance of MANETs protocols and schemes, assisting in the diffusion of information throughout the network. Grossglauser and Tse [70] produced the first work which pointed out that mobility can improve the network capacity, thus being a positive factor. As an example, Shah *et al.* used this concept to provide a service of collection of sensory information [32]. Thus, in this work, the network mobility is not considered a drawback, but instead will be very useful to extend the content dissemination to other geographical areas.

According to Gerla *et al.* [47], a high density of vehicles favors the P2P schemes since it promotes a higher exchange of content pieces, leading to a faster delivery. However, if this density is too high, additional problems related to network congestion might appear.

The majority of content distribution schemes is based on broadcasting in order to achieve a higher reachability. Since radio signals are likely to overlap with others in the neighborhood, a straightforward broadcasting by flooding is usually very costly because it will result in serious redundancy, contention, and collision, which is the *broadcast storm* problem. This problem is particularly relevant in high density networks (e.g. parking lot of vehicles). Tseng *et al.* [71] identifies this problem and proposes several schemes to address it in a MANET.

3.3.3.2 Piece and Peer Selection

According to Legout *et al.* [72], the piece and peer selection are two key factors in a P2P content distribution scheme. In a P2P scheme the content under dissemination is divided into smaller pieces that are exchanged among vehicles, these being clients and servers at the same time, whereby they can send and receive any piece to and from any other network node (peer). The piece and peer selection assumes a crucial role, whereby an efficient piece selection strategy is mandatory to enhance a successful dissemination.

In terms of the peer selection, if the communication is performed in broadcast and all the vehicles are able to act as relay nodes, this selection has a minor importance since every node tends to broadcast information to all of their neighbors. However, as seen in the REDEC work [55], an efficient peer selection could maximize service capacity of the system, along with keeping the network resources usage as low as possible.

As seen in a large amount of P2P schemes, the rarest first algorithm is the chosen piece selection strategy. This is the strategy employed by BitTorrent, which consists of the selection of the rarest piece to be disseminated first. According to [73,74], the rarest first algorithm presents a better performance than random piece selection strategies. On the other hand, Gkantsidis and Rodriguez [62] performed a simulation study which concluded that the rarest first policy could lead to a scarcity of certain pieces of the content under dissemination, whereby they proposed an approach based on network coding, which has the already stated potential advantages and enhancements.

3.3.4 Summary

This section displayed an overview of several content distribution schemes in a vehicular environment. There are P2P schemes which only use V2V communication to distribute the content and others that also use the fixed-infrastructure performing a V2I communication. Moreover, several schemes which use network coding concept to enhance their performance were also presented. The use of other types of technologies such as cellular (3G/3.5G/4G) and Wi-Fi is also important when the content has highly restricted requirements concerning its dissemination.

The great majority of the P2P schemes presented relies on *gossiping*. As stated by Gerla *et al.* [47], gossiping has a large set of advantages. This concept works efficiently in vehicular environments characterized by intermittent connectivity, and where the content is spread hop-by-hop across the network vehicles through opportunistic contacts until it reaches the final destination. However, this gossiping model can lead to large delivery delays, which are only acceptable for the dissemination of non-urgent contents.

In order to overcome the previously mentioned and other challenges (e.g. peer selection and high packet overhead), the concept of network coding was used in several content distribution schemes. As stated by Ahmed and Kanhere [65], the randomization introduced by network coding enhances distribution efficiency. Several works have been developed in the last years, and in the majority of the cases their performance outstrips the P2P schemes.

However, some contents require a higher QoS during this dissemination. Thus, several works proposed the use of other technologies to accomplish those goals. The most used is the cellular technology, especially the last generations (3G/3.5G/4G), since they have higher throughputs and their design is focused on data communications. On the other hand, cellular technology uses point-to-point communication channels which can lead to high congestion if a large number of vehicles are connected to the same repeater. Moreover, the higher cost of these cellular technologies when compared to the DSRC make them less desirable for non-urgent content distribution scheme.

A set of key factors that must be addressed in a content distribution scheme design are also identified. The most relevant are the network mobility and density patterns, and the piece and peer selection.

3.4 Simulation

As in many other networks, vehicular network simulation is essential to develop and test new protocols and systems before advancing into real-world deployment and experimentation. Wherefore, according to [75,76], in order to minimize the gap between reality and simulation and the number of experimentations, the simulation mechanisms must be as accurate as possible.

In this section two key components of VANET simulation will be analyzed: Mobility Models and Network Simulators. At the end, DTN and VANET simulators, specially built for vehicular environments will be described. A more complete survey of this topic is available in [77].

3.4.1 Mobility Models

A key characteristic of VANETs is the mobility of nodes. Therefore, an appropriate mobility model is needed to perform an accurate simulation of these networks. According to [78,79], vehicular mobility models can be classified as *macroscopic* and *microscopic.*

Macroscopic models describe the general properties of mobility, such as road topology, vehicle density, speed limits, number of lanes, traffic patterns, etc. On the other hand, microscopic models aim to give a detailed view of mobility, considering each car as a distinct element. Thus, according to Toledo [80], the car behavior is modeled in a more detailed way and is dependent on the state of its neighboring vehicles and the driver's characteristics.

Traditional MANET mobility models are not suitable for VANET simulation, whereby new mobility generation tools were developed specially for vehicular environments. These models are inserted as input parameters in network simulators, to model the vehicles' mobility. Below a set of existing mobility models are presented and at the end of this section a critical analysis of these will be made.

Simulation Urban MObility (SUMO) [81,82] is one of the most popular mobility simulators for VANETs. It is an open-source traffic simulator which allows modeling of inter-modal traffic systems (private and public transportation, and pedestrians). In addition to that, it is a highly-portable solution which uses a microscopic mobility model to handle large road networks with low processing requirements.

Development of Inter-VEhicular Reliable Telematics (DIVERT) [83] is a microscopic simulator which works with real maps. It was developed for V2V network simulation in urban scenarios and considered two types of vehicles: *sensors*, vehicles with communication capabilities, and *vehicles*, which do not have any communication feature and are just moving. It is important to take into account vehicles which are not communicating since they introduce challenges to communication, such as an obstruction to signal propagation.

VisSim [84] uses a microscopic model which includes car-following and pedestrian mobility models. Moreover, it has a very powerful Graphical User Interface (GUI) which allows the design of maps and simulation scenarios.

PARAllel MICroscopic Simulation of road traffic (PARAMICS) [85] is a scalable traffic generator tool designed for a large variety of environments, from an isolated intersection to a congested highway. It produces a microscopic model of the scenarios. It is used in more than 80 countries by governmental agencies, academic researchers, commercial consultants, and transportation companies and manufacturers.

VANET Mobility Simulation Environment (VanetMobiSim) [86] is based on CANU Mobility Simulation Environment (CanuMobiSim) [87] architecture . It is an open-source mobility generator, specially designed for vehicular environments which provides a set of interesting features for vehicular communication, such as specific speed limitation in certain roads. Furthermore, it produces detailed vehicular movement traces using macroscopic and microscopic models and allows the customization of simulation scenarios.

3.4.2 Network Simulators

There are a large set of network simulators available for a variety of purposes. In this section the most popular network simulators are described, giving a special emphasis to the ones suitable for VANETs or/and DTNs.

3.4.2.1 General Simulators

QualNET [88] was developed at the University of California and is maintained by Scalable Network Technologies. It is a commercial version of GloMoSim [89,90]. It provides accurate wireless simulation models based on Bit Error Rate (BER). Qualnet's mobility models are rather limited even though they provide a significant set of propagation models which include a specific designed model for VANETs called CORNER [91]. Moreover, it has a powerful GUI and is a scalable solution being able to simulate scenarios with thousands of nodes.

According to [92], OMNeT++ is a modular, extensible, component-based C++ simulation platform and library, primarily for building any kind of network simulators, either computer networks or any other. Since it is an extensible framework, specific modules can be added to the initial framework making it suitable for a specific network, such as a VANET. OMNeT++ already has a set of additional libraries for specific networks. The MiXiM [93] is a simulation framework for mobile and wireless networks which uses the OMNeT++ simulator. It offers detailed models of propagation, interference estimation, power consumption and MAC protocols, supporting vehicular communications.

NS-2 [94] was initially developed by United States (US) Defense Advanced Research Projects Agency (DARPA) through the Virtual InterNetwork Testbed (VINT) project. This simulator has been, for a long time, the elected choice for academic research. It is an open-source solution implemented in C++ and Tool Command Language (TCL) which covers a very large number of applications, network types and elements, protocols, and traffic models. However, it does not have any radio propagation model suitable for VANETs communication. Regarding this, Gukhol and Cherkaoui [17] developed an implementation of IEEE 802.11p to NS-2 simulator. Due to its complexity the development of NS-2 was discontinued in 2010.

NS-3 [95] is an evolution of NS-2 and it recently turned into the most adopted network simulator by the research community since its code documentation has been well organized facilitating the use of it. NS-3 relies on C++ and Python for the implementation of new simulation models. Moreover, TCL, which was one of the sources of complexity in NS-2, was eliminated. In contrast to NS-2, NS-3 provides a wireless model based on BER being suitable for wireless communication simulation. Weingartner *et al.* [96] conclude that NS-3 has the overall better performance among a set of widely used network simulators.

3.4.2.2 VANETs simulators

As mentioned before, a variety of mobility and network simulators are available, however, to perform an accurate simulation of a vehicular network, they need to be assembled together. In the following paragraphs the most significant integrations of these two important pieces [14] are introduced.

Traffic and Network Simulation Environment (TraNS) [97,98] integrates network and mobility simulators (SUMO [81] and NS-2 [94]) to simulate a VANET behavior. It was the first simulator to create a feedback loop between mobility and networks simulators. It comprises two modes of operation: *application-centric* and *network-centric*. The application mode, the network simulator is able to influence the mobility simulator. In the second mode, the network simulator just parsed mobility logs to perform its operations. Since 2008 that TraNS is not maintained, although it was very important in the past and established a turning-point in VANETs simulators.

VEhicles In Network Simulation (VEINS) [99] integrates SUMO and the INET framework [100] of OMNeT++ [92] simulator. Both components work in parallel to perform Inter-Vehicle Communication (IVC) evaluation and communicate through the Traffic Control Interface (TraCI) [101] allowing bidirectionally-coupled simulation of road and network traffic. VEINS is an open-source solution which relies on a trusted vehicle mobility model and on fully-detailed model for vehicular communications layers (IEEE 802.11p and WAVE). Furthermore, it allows the introduction of models to modulate the shadowing effects caused by buildings as well as by vehicles. The simulator was validated through a set of experiments using different V2V communication protocols.

iTETRIS [102] is an European project that joins SUMO [81] and NS-3 [95]. The architecture of iTETRIS is completely modular and flexible wich allows the integration of a variety of mobility and network simulators trough open Application Programming Interfaces (APIs). The iTETRIS Control System (iCS) is the central module responsible for the coordination between both simulators. The applications are implemented in an external block on top of iCS module, whereby platform users can create their applications in a "language-agnostic" way.

National Chiao Tung University Network Simulator (NCTUns) [103] is different from the previous simulators since it does not relies on a junction of two different simulators (mobility and network), but it builds everything from scratch. The main goal is to create a network simulator capable of acting as an accurate mobility simulator for vehicular environments. According to Wang and Li [103], NCTUns supports the simulation and emulation of a VANET which uses the IEEE 802.11p and WAVE standards. Furthermore, it supports distinct vehicle mobility models, road network construction, simulation and emulation of OBUs or RSUs, and communication using a variety of protocols such as IEEE 802.11b (operating in a infrastructure or ad-hoc mode).

3.4.2.3 DTNs simulators

The routing and network protocols specially developed for DTNs need to be evaluated whereby DTN simulators were created since the real-experiment requires the deployment of a testbed which can be somewhat flexible, expensive, and limited. To evaluate the performance of such protocols, the simulator needs to include node mobility in its execution. According to [104], the source of this mobility could be (a) synthetic mobility models, and (b) obtained from real-world measurements. A variety of projects have been collecting traces (nodes position, contacts duration and time, and peers) of contacts between network nodes [105–107]. Such information constitutes a valuable source of information for validating and improving the characteristics of synthetic models. However, realistic data can be quite limited in number of nodes and coverage area whereby sometimes the use of a model-based synthetic mobility generation is more useful since it could be more flexible and scalable (see section 3.4.1).

DTNSIM [108,109] was developed by Jain *et al.* to compare the performance of routing protocols in a DTN. It is a Java-based implementation working as a discrete event simulator. In this simulator the nodes have limited storage capacity and it can create and destroy nodes and links dynamically, temporary, or permanently. The links are modulated as attached to nodes performing a directional communication, with a finite propagation delay and bandwidth. The availability of a link can also be controlled randomly or by specifying in a file a certain up time for a certain link.

The Opportunistic Network Environment (ONE) simulator [104,110,111] is a Java-based simulator specifically designed for evaluating routing and application protocols in a delay-tolerant environment. Most of the DTN simulators focus only on routing simulation. However, the ONE simulator combines a set of functionalities such as routing, mobility modeling, notions of energy consumption, visualization and analyzing modules, and statistical reports. The ONE simulator already has a set of default routing protocols: Direct Delivery, First Contact, Spray-and-Wait (SaW) [42], Probabilistic Routing Protocol using History of Encounters and Transitivity (PRoPHET) [112,113], Epidemic [39], and MaxProp [40]. Furthermore, it also defines three different mobility models: random movement, map-constrained random movement, and human behavior based movement.

3.4.3 Summary

As was seen, there is a wide range of mobility models available, all of them contemplating different features and distinct levels of detail. Although the reviewed mobility models aim to replicate an accurate model of the network behavior, it is always an approximated model, not being an exact replication of it. Moreover, the more detail a mobility model encompasses, the more processor hungry it will be, whereby it is a crucial factor on the simulation procedure.

In this work we have access to a full dataset of information which contains mobility and network data of two 24-hour periods. Information about the vehicles' location, speed, direction of movement, or its neighbors are stored in those datasets. Thus, there is no need to resort to a mobility model which would be responsible for the input of vehicle mobility to the network simulator. The collected information is used as an input to the platforms used to evaluate the proposed content distribution strategies in Chapter 6.

There are already a large set of network simulators available which contemplate specific solutions for generic, delay-tolerant, and vehicular networks. Despite there being this wide array of simulators, it was decided not to use any of these. This decision is due to the pre-existence of a Matlab emulator created by IT and Veniam® which emulates the upload process of information from remote sensors to the fixed infrastucture of the vehicular network, assuring the transport of information through the vehicles. Thus, as described in Chapter 5, this already developed emulator was subverted to emulate a content distribution scenario where mobile nodes collect data from the fixed infrastructure and spread it out through the vehicular network using vehicles to carry this information. Moreover, since the proposed content distribution strategies must be implemented in the Helix DTN solution, a new emulator was developed, since none of the described simulators are able to exactly recreate the behavior of this new solution. This emulator is described in Chapter 5 and several improvements and features are added under the scope of this work.

The mobility of the previous emulators is introduced through the use of the real data collected and stored in the previously mentioned datasets.

3.5 Delay Tolerant Networks' Implementations

After regarding the theoretical concepts associated with vehicular networks and delay tolerant networks, this section presents the results of a survey on real implementations of the Bundle Protocol [28]. The following subsections briefly introduce the implementation, outlines design features, and highlight some of the characteristics of each implementation relevant to this work. First the general implementations suitable for use in the large part of DTNs are introduced. After that, several implementations specifically developed for a vehicular environment are presented and discussed. In this section is also described a new implementation of a DTN for a vehicular environment that is used as the selected implementation during this document.

3.5.1 Widespread Solutions

The Bundle Protocol was defined by Scott and Burleigh [28]. These researchers developed their work in the Delay Tolerant Network Research Group (DTNRG), which is a research group from the Internet Engineering Task Force (IETF). The main goal of this research group is the study, development and implementation of architecture and protocols to operate in challenging environments where continuous connectivity is not ensured. Within this context, they developed the Bundle Protocol, which implements an overlay network allowing the connectivity and communication in those challenging environments.

Several implementations of the Bundle Protocol were developed. The most used by the research community are presented in the following sections.

3.5.1.1 DTN2

DTN2 [114] is the reference implementation of the Bundle Protocol proposed by the DTNRG which provides a flexible framework for DTN experimentation and real-world deployment.

The system architecture of DTN2 is illustrated in Figure 3.8. The bundle router module is the major component of the implementation. It requires several information about the state of the system to perform routing decisions. After the decisions are made, the router sends a set of instructions to the forwarder which is responsible for executing the actions. As such, DTN2 separates the control from execution which allows easy extension, modification and replacement of the router module. The functionalities and specifications of each module are described as follows.

The **Bundle Router** is responsible for making routing decisions (e.g. select a route). It makes its decisions based on external events, which affect routing decisions, and passes a set of instructions to the **Bundle Forwarder** who is responsible for the execution of routing decisions. To execute these instructions, the forwarder interacts with the storage and the convergence layers. Separating the bundle forwarder from the router allows the

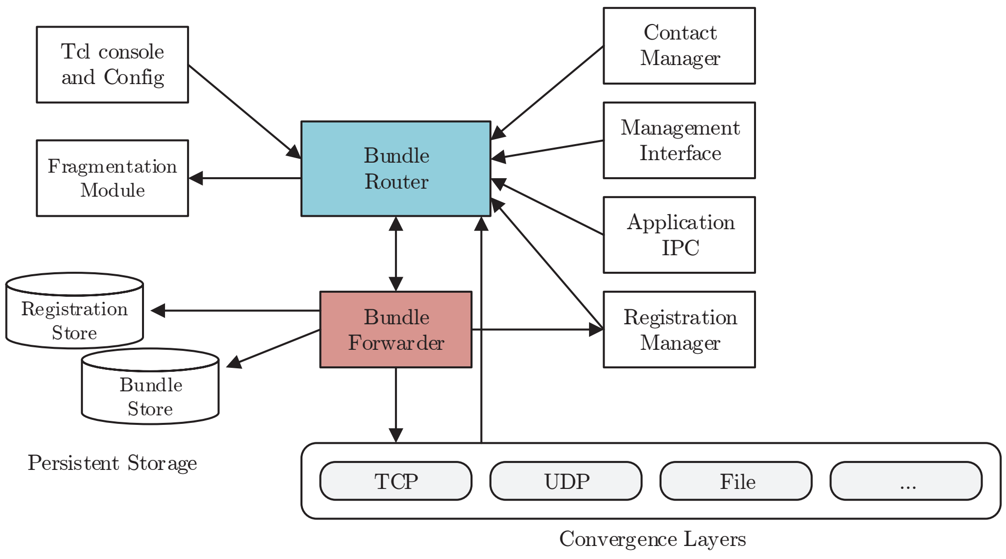


Figure 3.8: DTN2 system architecture, based on [114]

implementation of several routing policies, and the separation of the calculation of instructions from their execution providing an isolation of the routing code from changes in other internal APIs.

Convergence Layers are adapters between Bundle Protocol and underlying transport and network layers (e.g. TCP or User Datagram Protocol (UDP)). They allow the conversion of bundles into proper data streams capable of being transported by underlying transport protocols from one hop to another.

Persistent Store is composed by two databases which are responsible for storing the content of bundles during the carry phase of the SCF mechanism.

Fragmentation Module is the module responsible for the fragmentation and further reassembling of bundles. When all the fragments of a certain bundle have been received, this module signals the router.

Contact Manager is responsible for keeping an updated record about available links. This information includes historic data about their connectivity and performance, and also information about possible future contacts. After converting this information into an abstract contact description, the router module uses it to perform routing decisions.

The DTN2 implementation provides a set of routing protocols such as PRoPHET [112, 113,115], Delay Tolerant Routing for Developing Regions (DTLSR) [116,117], flood [118], tca-router [119], external [120] and static [121].

3.5.1.2 ION

Interplanetary Overlay Network (ION) [122] is the Bundle Protocol implementation proposed by Jet Propulsion Laboratory (JPL) specially developed to operate on spacecrafts in an inter-planetary environment. It was designed to support high-speed, small-footprint deployment of DTN in embedded systems.

The design of ION is similar to a database and the persistent storage is based on the Simple Data Record (SDR) which already exists in spacecrafts. The SDR module allows storing on disk, in memory or in both. It also ensures the data integrity in case of database failure.

The concept of shared-memory is used to design the all system architecture but also to establish communication between sender and receiver processes. In ION, there is no discovery of neighbors, the contacts are scheduled. In space environments the available bandwidth is small whereby the ION system is optimized for this characteristic, supporting Compressed Bundle Header Encoding (CBHE) [123] and Licklider Transmission Protocol (LTP) [124]. In addition, it also implemented the Bundle Streaming Service (BSS) [125] for streaming audio and video over a DTN.

3.5.1.3 IBR-DTN

DTN2 is the reference implementation adopted by DTNRG, but unfortunately it is not suited for embedded systems. This fact precludes its use in networks composed by devices which have constraints in terms of performance and energy. To fill this gap, the Institut für Betriebssysteme und Rechnerverbund (IBR) of the Technische Universität Braunschweig (TUB), in Germany, developed a new implementation of the Bundle Protocol called IBR-DTN [126].

IBR-DTN was specially developed for embedded systems and runs in OpenWRT [127] which is a Linux-based Operating System (OS) created to run on this type of hardware. Nowadays, it runs on a variety of systems and OSs, such as Windows, Mac OS, or even Android.

Figure 3.9 shows an architectural overview of IBR-DTN. According to [126], the implementation of IBR-DTN follows two guidelines. The first one is to minimize the external requirements (e.g. libraries). This guideline is due to the fact that sometimes these external dependencies are not available for certain platforms, or the disk/memory space needed for them is prohibitive. This feature also allows the easy exportation of IBR-DTN to other platforms. The second guideline was to keep the software as modularized as possible. Therefore, the IBR-DTN was tailored to the capabilities of the used platform.

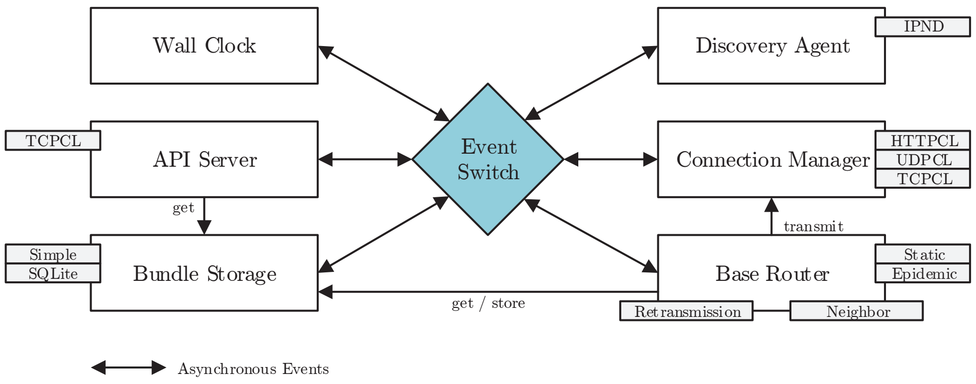


Figure 3.9: IBR-DTN system architecture, based on [126]

According to Schildt *et al.* [126], the architectural design of IBR-DTN is divided into several modules which will be presented and discussed below.

Event Switch is the core module of the IBR-DTN implementation, and it is responsible for dispatching raised events to all proper sub-modules. A high level of competition among the modules can be achieved due to the queuing of events to a private work queue of module's thread. Any module (existing or new) can raise or receive an event to communicate with other modules. The architecture allows the creation of new applications by any standard module. The standard implementation already has events related storage, routing or neighboring.

Discovery Agent is the module responsible for the discovery of neighboring nodes. It performs the discovery implementing the DTN IP Neighbor Discovery (IPND) version one or two as specified in [128], and IP-discovery frames compatible with DTN2. The *Discovery Agent* is the entity which implements the DTN IPND. This entity controls the appearance and disappearance of neighbors and, when one of them occurs, it generates an event reporting it. This event is heard by the Base Router module and, if there is any bundle for the discovered neighbor, the node sends it.

Connection Manager is responsible for interconnecting the CLAs with IBR-DTN internal modules. As mentioned before, the lower protocols used to establish a connection among DTN daemons are called CLAs. In the IBR-DTN architecture these elements are implemented as modules, and each one of these modules provides an interface to transfer (and receive) bundles to (from) other nodes. The transfer and reception processes generate global events and, in case of reception of a new bundle, it is stored in the Bundle Storage. The IBR-DTN has four different convergence layers:

- *TCP Convergence Layer*: this convergence layer is compatible with [129]. It uses a handshake mechanism between daemons and can split bundles into segments, which are then acknowledged by the receiving node;
- *UDP Convergence Layer*: this convergence layer is compatible with [130]. To fit in the UDP requirements the size of the bundles used within this convergence layer have a maximum size equal to the UDP datagram maximum size;
- *Hyper Text Transfer Protocol (HTTP) Convergence Layer*: this convergence layer is based on libcurl [131] and can use a HTTP server to send and receive bundles;
- *Low Personal Area Network (PAN) Convergence Layer*: this convergence layer supports the IEEE 802.15.4 MAC protocol [132] which is used in various WSNs.

To implement the characteristic SCF mechanism of DTNs a storage module is required, the **Bundle Storage**. This module has to be able to store bundles for long periods of time. When the routing module wants to peek a bundle to send, it queries the storage asking for a specific bundle. Within the querying the bundle can be identified by multiple parameters as its unique ID or its destination. The IBR-DTN offers three different types of storage:

- *Memory*: this is a non-persistent storage and it is chosen by default when no storage path is defined. In this case, bundles are stored in Random Access Memory (RAM), and the maximum possible number of bundles stored is limited by the OS;
- *File based storage*: this is used when a storage and path are defined. In this case, bundles are stored persistently (e.g. in a hard disk), and remained stored even if the device is turned off. Since the memory is no longer used to store bundles, the device can use all the memory for system processes (e.g. routing, neighbor discovery, etc);
- *SQLite*: this type of storage relies on a SQLite database [133] and can be useful for more complex routing modules.

Bundle Router is responsible for managing the interaction between the implemented routing modules with the IBR-DTN standard modules (Bundle Storage, Connection Manager and Discovery Agent) to perform routing decisions. It receives information about new incoming neighbors, sent by Discovery Agent and, when a node wants to send a bundle, it contacts the Connection Manager and request the correct CLA to tranfer the information. IBR-DTN offers several different routing sub-modules:

- *Static*: all the routes and paths are configured at the beginning of execution and it is assumed that they are always available.

- *Neighbor*: information is sent to all the neighbors discovered by the Discovery Agent (at the time of decision).

- *Epidemic*: this sub-module implements an Epidemic Routing [39]. The major difference introduced by the IBR-DTN implementation is the use of a BloomFilter [134] mechanism instead of summary vectors. In addition to that, it also propagates a "purge" vector through the network to inform the nodes which bundles can be deleted from storage.

- *PRoPHET*: this sub-module implements the PRoPHET routing protocol [112,113].

- *Retransmission*: it is responsible for signaling errors occurred during the transmission of a bundle. When a transmission error occurs, the bundle is re-queued and remains in storage for further retransmission. There are two types of errors, permanent and temporary. A bundle is retransmitted only if the error is temporary.

Wall Clock establishes a global clock to be used by IBR-DTN. This module provides a global time tick event used by Event Switch to dispatch the events in the other modules.

The IBR-DTN provides a socket-based API, the **IBR-DTN API**, reusing the bundle streaming protocol. The interface can be a TCP-socket to establish communication through different machines, or a Unix Domain Socket (UDS), to be used locally. The IBR-DTN provides a library to be linked to applications in order to simplify the creation of bundles. As such, all supported DTN features of the daemon are immediately ready. The implementation does not support out-of-band messages which invalidates real-time configuration.

Such as DTN2, IBR-DTN also provides a set of applications for testing and debugging. They are summarized in Table 3.2.

3.5.1.4 Others

There are other implementations but these are less widespread compared with the previous. For example, the POSTELLATION [135,136], JDTN [137], Bytewall [138], or DT-Talkie [139].

3.5.1.5 Comparison of implementations

Table 3.2 summarizes the previous characteristics of the widespread implementations previously described. It was added to the table information about less used implementations such as POSTELLATION [135,136], Bytewall [138], and DT-Talkie [139].

Table 3.2: DTNs widespread solutions comparison

DTN Implementation	**OS**	**Programming Language**	**Security Support**	**Applications**	**Routing**
DTN2 [114,140,141]	Linux, MacOS X, Solaris, FreeBSD and Linux on ARM	C++	OpenSSl and partial support for BSP [142]	dtnping, dtnsend, dtnrecv, dtncp and dtncpd	Static, Epidemic [39], Flooding, PRoPHET [112,113], DTLSR [116], TCA [143], external routing via XML
ION [122,144,145]	Linux, MacOS X, Solaris, FreeBSD, VxWorks, Windowns and uClibc	C	(not specified)	Space flights ans support for Bundle Streaming Service [125]	CGR [146]
POSTELLATION [135,136]	Windows, MacOS X, Linux, *BSD and RTEMs	C	TCP over Transport Layer Security (TLS) support	dtnping, dtnpong, dtnsend, dtnrecv, HTTP/HTTPS proxy and video streaming	(not specified)
IBR-DTN [126,147,148]	OpenWRT, Debian/Ubuntu, Debian ARM, MacOS X, Gentoo Linux, Windows and Android	C++	Combination of 4 levels (none, authenticated bundles, encrypted bundles, signed bundles) based on [142]	dtnsend, dtnrecv, dtntrigger, dtnping, dtntracepath, dtninbox, dtnoutbox and dtnstream	Static, PRoPHET [112,113], Epidemic [39] and Flooding
ByteWalla [138]	Android	Android SDK 1.6	Support for Bundle Security Protocol (BSP)	dtnping and e-mail	Static and PRoPHET
DT-Talkie [139]	Maemo based Nokia Internet Tablet, Symbian, MacOS X, Linux and Openmoko	Symbian OS SDK	(not specified)	Practical voice communication	Flooding

3.5.2 Helix

3.5.2.1 Introduction

In previous works developed in the Network Architectures and Protocols (NAP) [149] research group several problems were identified when implementing a DTN software in a vehicular environment. Tavares [150] implemented and tested two implementations: DTN2 [114] and IBR-DTN [126] and he detected a set of problems. The author noticed that DTN2 had a bad implementation of the PRoPHET and flood routing protocols. Furthermore, he concluded that DTN2 is not robust in high mobility environments precluding its use in VANETs. Thus, he implemented and ran a few tests successfully in IBR-DTN implementation. As a continuation work, Guedes [151] successfully tested the IBR-DTN in a large-scale vehicular network. However, he concluded that IBR-DTN introduces unnecessary complexity for intended applications.

Regarding this, at the end of 2014, a partnership between IT NAP [149] and Veniam® started the development of a new implementation of the Bundle Protocol. The new implementation, called Helix, was specially designed to operate in VANETs and its main goal is to reduce the complexity associated with other implementations (e.g. IBR-DTN). This is a proprietary solution , whereby only a brief explanation of its architecture and functionalities will be given in the following subsections.

The Helix software is developed in a C/C++ programming language and was designed to be highly modular and extensible. It supports communications using the reference standard IEEE 802.11p and WAVE protocols, and further the conventional Wi-Fi technology, IEEE 802.11a/b/g.

3.5.2.2 Architecture

An overview of Helix architecture is described in Figure 3.10. As illustrated, it is composed by seven modules: Neighboring, Socket, API Management, Storage, reception (RX), and Routing. Each one of them is responsible to implement and execute a set of functions necessary to the operation of a DTN. In order to run their functions, modules can interact with each other through Inter-Process Communication (IPC) sockets, and a node can exchange packets with other nodes using the UDP transport protocol.

Communication

The **Socket** and **RX** modules can be included within the communication since they work in tandem to process incoming and outcoming packets of data or control (e.g. neighboring messages).

The Socket module is an abstraction layer to send/receive packets to/from neighboring nodes, and it manages the access to a UDP socket.

The RX module has an internal thread which is constantly checking if any data was received in the UDP socket. When it occurs, the RX module analyses and classifies the packets according to its flags (e.g. Neighbor Acknowledgment (ACK), End-to-End (E2E) ACK, etc.), destination EID (endpoint or relay), and service ID (control or data). After this classification, the module forwards the packets to the routing module (data packets), or to the neighboring module (neighboring control packets).

Neighboring Discovery

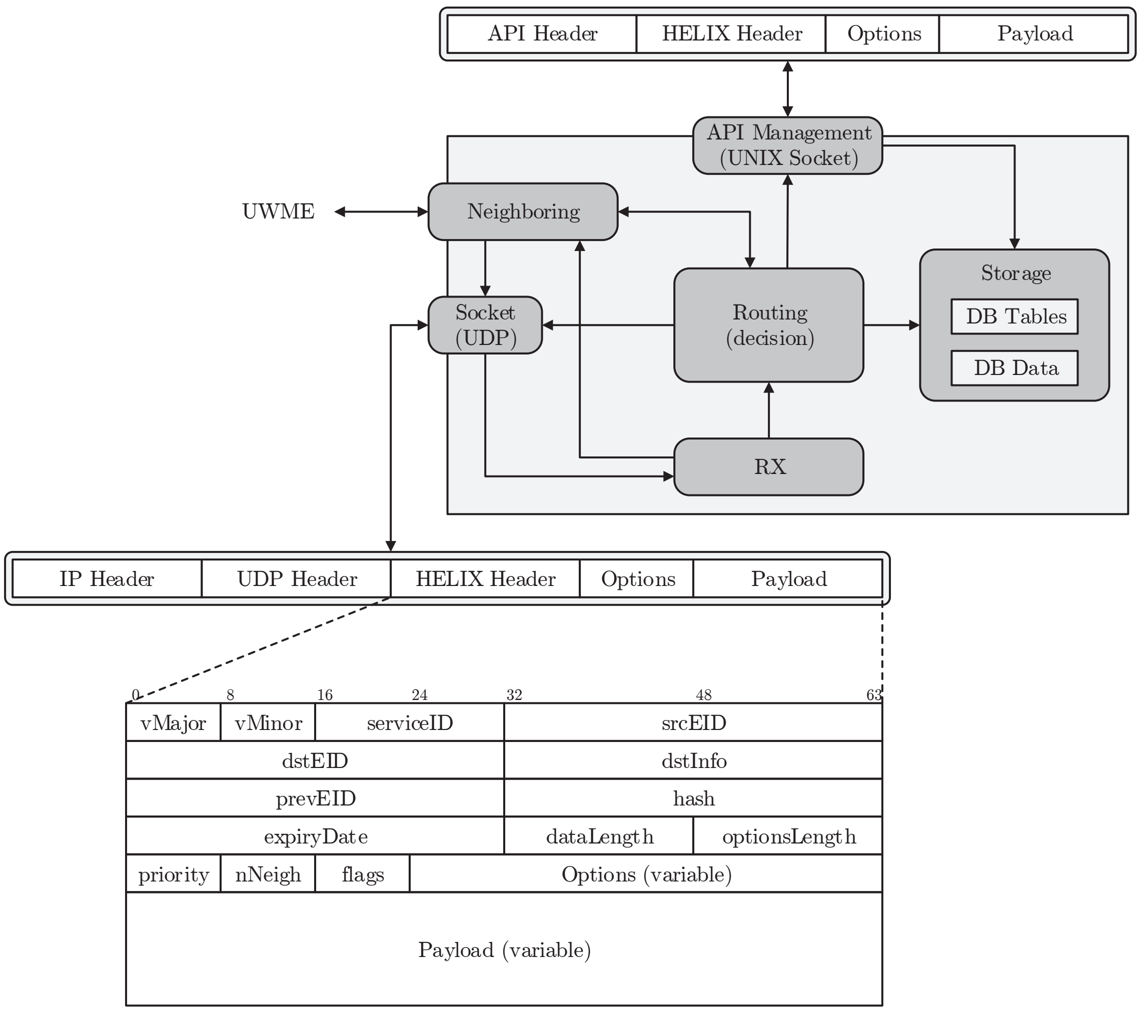


Figure 3.10: Helix architecture

Analogously to the Discovery Agent module in IBR-DTN, the **Neighboring** module is responsible for the discovery of neighboring nodes. It can operate with different types of neighboring nodes depending on the communication interface that has been used. It supports communication through Wi-Fi, WAVE, and Ethernet interfaces. Furthermore, it defines a new type of neighbors as Static to deal with predefined static routes between nodes. Since it is developed for vehicular communications, the WAVE neighbors are OBUs or RSUs, the nodes with Wi-Fi interfaces are typically sensors or endpoints, and the RSUs and servers located in the core network are connected to an Ethernet interface or a static link.

The initialization of the Neighboring module is done by the Routing agent (at its creation). After that, it performs a periodical search for new neighbors. Each node sends a Neighbor Announcement packet advertising its presence. Upon receiving such packet, a node updates its internal neighboring tables with a set of information: EID, IP address, type (e.g. RSU or OBU), communication port, and Received Signal Strength Indicator (RSSI) - specific of WAVE neighbors.

The developers faced a set of challenges in order to maintain updated neighboring tables of multiple interfaces from different technologies in order to minimize the routing wrong decisions. Their main goals were to minimize flooding messages in the network and the consumption of resources (e.g. CPU), and to exploit the advantages of WAVE brought by the WME (see section 2.2.4) such as service announcements. Thus, the selected strategy was the creation of one class for each type of Neighboring with its own thread (see Figure 3.11).

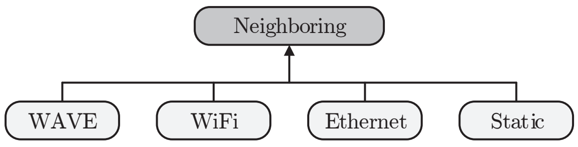


Figure 3.11: Neighboring classes

A set of methods was created in each class to update the internal neighboring table or access their information. In the WAVE class, methods were developed to register/deregister a Provider Service (PS) for Helix, check for the communication channel, and create one if there is none available, get network information (e.g. IP address and network mask) of WAVE interfaces, and get information about a specific type of neighbors (e.g. only OBUs ou RSUs) or a certain EID. In the Wi-Fi class, the methods created also allow to obtain information about a specific neighbor or a set of them, check the network information related with this interface, and manage the thread to send periodically Neighbor Discovery messages in broadcast (if node is an OBU), reply to Neighbor Announcements, and update internal tables. In the Ethernet class, methods were also created to get information about neighboring nodes, update internal neighboring tables, and retrieve network information of this interface. However, these methods are only applicable to RSUs and local servers. The methods in the Static class only maintain a list of Neighbors from the configuration file (e.g. remote servers).

Persistent Storage

The **Storage** module is responsible for storing several packets and other information that is relevant for the forwarding decision. The development of this module aims to fit a set of requirements to do not compromise the performance and transfer opportunities, and minimize the packet losses, had to be robust in order to deal with unexpected power outages, allow a binary storage of network packets, and perform fast queries to accommodate routing decisions.

The persistent storage module is responsible for holding the data psackets and the additional information required to handle and forward them. This module is in the center of all the activity happening around a node - it can be queried by the Routing module regarding the existence of a packet in Storage, and if the packet is new, receive it. At the same time, an application can push a packet to the Storage that is to be sent to another node, and the Routing is also performing its algorithm of packet forwarding.

All this can happen concurrently, as some blocks are running their own threads and there is no synchronism between them. Therefore, it is very important that the Storage is able to handle all these processes in an efficient way. Every public operation is thread-safe and that means that threads may be blocked while some other operation is occurring.

As illustrated in Figure 3.12, the Storage module contains two sub-modules: *StorageDisk* and *StorageRAM*; and its logical organization can be divided into two internal sub-blocks: *StorageData* (contained in RAM+Disk) and *StorageInfoTables* (only contained in RAM). StorageData manages the persistent storage of packets on disk, it writes/reads files to get packet contents (using a file per helix packet), and uses the StorageRAM sub-module to improve the performance of such operations. StorageInfoTables are in-memory tables with search optimization in order to perform easy and fast queries. The tables are implemented in the StorageRAM sub-module and four different organized tables are available: *Expiry* (order by expiry time), *OnHold* (order by time on hold left), *Own* (packets which are meant for this node and are ordered by serviceID), and *NoData* (table of packets known, but no data is stored on disk, they are ordered by expiry time). Furthermore, there is another table which organized the storage packets according to its identifier, called *hash*. Such tables allow easy and fast queries such as “Next packet to expire”, “Next packet to expire for destination EID=123”, or “Contains packet with hash=234?”, among others.

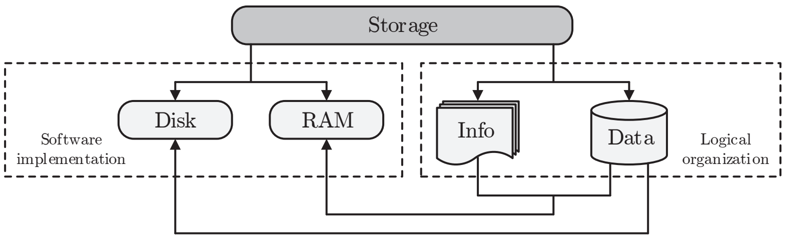


Figure 3.12: Storage organization

Routing

The **Routing** module is responsible for the decision of “which packets” should be sent to “what neighbor” at “what time”. The major challenges at the development were related to the maximization of the delivery of useful information to its destination and the sent information during transfer windows, minimization of the CPU consumption, and balancing of the load between nodes. Furthermore, it aims to minimize the replicas in the network and the packets maintained in Storage.

Helix adopted a hybrid routing solution since it routes per Neighbor and per Packet type. The first routing decision is based on the packet type (data or control), but the remaining process depends on the node’s type. Thus, the following class diagram (see Figure 3.13) was implemented. At the RX module the packet is already filtered based on its type, being forwarded to a specific part of the Routing module. In order to send a packet a node must check if it has any available neighbors and select one (or more) to send a packet that should be picked from the Storage module.

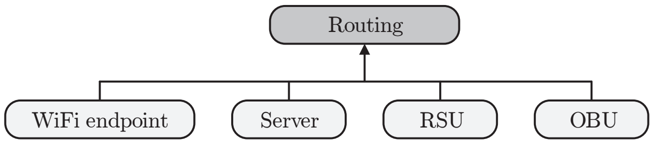


Figure 3.13: Routing classes

However, this module is still in a development phase, and only supports upload traffic flows from sensors or endpoints to local servers. Thus, the main application already implemented and tested in Helix is the collection of sensor data and its forwarding to the Internet for statistical treatment.

API Management

A Helix node interacts with external applications trough the **API Management** module. This module uses UNIX sockets Datagram Communications (connectionless) to manage data and control messages between Helix and Helix Apps. It was defined that IPC messages should be used to separate control from data messages.

This module has a thread to treat packets from Helix Apps through the API socket and creates an abstraction layer to send/receive Helix Packets to/from API. Furthermore, it manages the access of Helix Apps to Helix (registration and deregistration). It has an auxiliary thread responsible for pulling own packets from Storage module.

A set of Helix Apps are already developed and tested such as HelixPing, HelixSendString/HelixRecvString, HelixSendFile/HelixInbox, and HelixMonitor/HelixCollectMonitor.

3.5.2.3 Operating Flowchart

As illustrated in Figure 3.14, Helix is a multi-thread software composed by multiple modules operating concurrently. Before the launching of these threads, the Helix is configured. This procedure analysises the configuration file and, according to its content, defines all the relevant parameters to run Helix as specified in this file. Along with other parameters, the configuration file specifies the port of the socket module, storage capacity, APIs, log and storage paths, interfaces of communication, or version of routing. All of this information is used in the procedure of creation and initialization of all the modules (threads) created in Helix. Thus, as the configuration file is read, the modules are created and initialized.

The creation of each object/module is associated with the launch of a new thread responsible for executing the behavior of this module as was designed. Since the moment that each thread is initiated they work concurrently to access the shared resources of the machine/platform on which they are running. This strategy makes Helix a suitable solution for platforms where the usage of resources is critical, such as a vehicular network.

Once all the threads are launched, the program runs until an external signal is sent. This signal is typically executed by the developer or user through a keyboard, and is interpreted by Helix as a command to stop all the threads and save the program context.

3.5.2.4 Packets Structure

The bundle layer is implemented above the transport layers, as suggested in the reference specification of DTN, Request for Comments (RFC) 4838 [2]. However, the Helix implementation does not strictly follow the reference specification of the Bundle Protocol described in RFC 5050 [28]. The main difference is related with the bundles, which in this case are called *Helix Packets*. The structure of the packet is as follows (see Figure 3.10):

- **HelixHeader**
  - Helix Version
  - Service ID (e.g. Neigh Discover or Content Distribution)
  - Source and Destination EIDs
  - Destination information (e.g. board version and node type)
  - Previous custodian EID

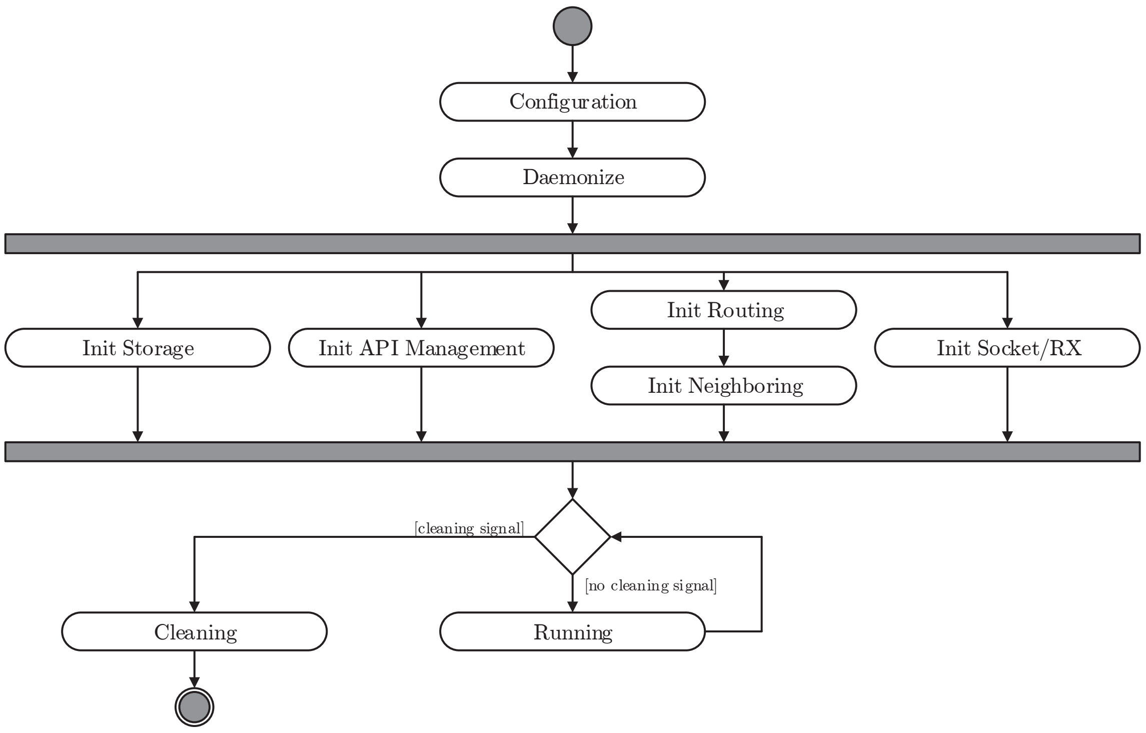


Figure 3.14: Helix operating flowchart

- Hash (unique identifier for a packet)
- Expiry Date (time of creation + lifetime)
- Payload Length
- Options Length
- Priority
- Current number of neighbors that received the packet
- Flags to identify the packets type (e.g. Neighbor ACK, E2E ACK, Delivered)

- **Options**: this is an optional field, options can be added later (e.g. list of neighbors where the packet was received)
- **Payload**: array of bytes with a maximum of 32 KB (if options were added the maximum size would decrease)

3.5.2.5 Applications

Helix was created mainly forecasting its use for two applications:

- **Collect Data**: Periodic Measurements from any Devices (e.g. OBUs logging/monitoring, sensors for garbage container status, environmental sensors, etc.)
- **Content Distribution**: Software Updates, Commercials/Advertisements, Entertainment Content (e.g. TV shows, daily news, popular videos, etc.)

3.5.2.6 Configuration

The Helix configuration is done using an auxiliary JavaScript Object Notation (JSON) [152] file which is read and parsed at the beginning of the execution. Several settings are configured in this file such as storage capacity and path, node EID, communication interfaces name, and static routes. An example of it is written below.

```
"\ac{EID}": "20001",
"socket_port": "4556",
"storage_path": "/root/DTN/Storage/",
"storage_cap": "3",
"api_path": "/root/DTN/api/",
"log_path": "/root/DTN/Log",
"log_output": "2",
"log_level": "2",
"11g_ifaces": [ "wlan0" ],
"11p_ifaces": [ "wlan1" ],
"eth_ifaces": [ "br-lan" ],
"static_routes": [
{
"Address": "192.168.7.111",
"\ac{EID}": "20001",
"Port": "4556"
}
]
```

3.5.3 Summary

In this section a set of DTN implementations were presented. First the widespread solutions are presented and described in order to take a brief look into the most used implementations. However, this first set of approaches is designed to be used in a wide range of devices and platforms, whereby their use is not optimized for specific environments like a vehicular network. In order to overcome this characteristic and tend to a specialized solution for the implementation of a DTN in a vehicular network, IT in a partnership with Veniam® developed a new solution called Helix.

Helix software was specifically created to be used in the FutureCities project testbed of Oporto city. As the main goal of this work is the implementation of several content distribution strategies using delay-tolerant mechanisms to spread non-urgent information, this new solution was selected to be the basis of this work. However, Helix is an initial implementation which needs to be improved in order to enhance a successful deployment of a content distribution service. Chapter 5 describes these improvements and modifications.

3.6 Chapter Considerations

In this chapter a considerable amount of previous and relevant work is presented and discussed. The following topics summarize the described related work as well as identify a set of key issues that are still not addressed and are overcome in this Dissertation.

Vehicular Delay Tolerant Networks

The VDTNs aim to overcome the intermittent connectivity and network disruption in a VANET through the implementation of delay-tolerant mechanisms. VDTNs bring the possibility of using V2V and V2I communications to bring a new set of applications, such as media and traffic information dissemination, along with the possibility of using a vehicle to collect information about the environment around it and disseminate that information through the network.

In this section several VDTN projects were presented and described. However, there is a lack of VDTNs projects in order to study and better understand the challenges and advantages brought by this kind of networks. Thus, as a large vehicular network is available under the scope of this Dissertation along with an already developed DTN solution for VANETs, this work aims to design and implement a content distribution scheme of non-urgent information using delay-tolerant mechanisms to run over this network. As shown in Chapter 6 this work correctly implemented a content distribution scheme to be used in the FutureCities project testbed, allowing the introduction of this new kind of services.

Content Distribution

In this chapter several content distribution schemes were presented. They were divided into three main groups: P2P, coding, and the ones that use multiple technologies.

Most of the P2P schemes use the gossiping of control messages to handle the peer and piece selection. They use this gossiping to advertise which pieces of the file they want to download. However, this work proposes a different approach. Instead of advertising the wanted pieces, the nodes periodically advertise their storage content. This approach gives the sender node the responsibility of evaluating which are the most lacking pieces within its vicinity. This change aims to speed up the initial content distribution since smaller messages are exchanged at the beginning of the process, allowing for a faster evaluation of the next packet to be sent. This approach is applied in the Local Rarest Bundle First (LRBF) and Local Rarest Generation First (LRGF) strategies (described in Chapter 4).

Network coding proposes to overcome several challenges related with a pure P2P scheme. The majority of the proposed schemes rely on random linear network coding in order to overcome the peer and piece selection challenge. However, the CodeCast proposed an approach which uses a new parameter called rank. This parameter allows the sender node to evaluate which are the most lacking generations within its vicinity, and select the packets to send according to them. Thus, during this work a similar approach is used when the LRGF is applied.

The network mobility and density were identified as critical factors for disseminating content. Although the mobility in a network could lead to the link's disruption and intermittent connectivity, it can be also an advantage since it increases the coverage area to the content be disseminated. Moreover, the intermittent connectivity can be overcome using delay tolerant mechanisms, as proposed during this work. A detailed analysis of the mobility and density impacts in the real network used for this work is performed in Chapter 6.

Simulation

Due to the logistics involved in real-world experimentation and similarly to other kinds of networks, VANET simulation is a crucial point in the initial design thinking of new features. Therefore, the simulation procedure must be as accurate as possible to minimize the gap between reality and simulation, which is crucial for an easier real deployment of these features. Several mobility models and network simulators were described in this chapter.

Regarding the mobility models, due to the availability of a robust dataset containing logging information about the network nodes' location, movement, and vicinity, there is no need to use an already deployed model. The collected data will be used to introduce the mobility of the network in the platforms used for evaluation of the proposed content distribution strategies in Chapter 6.

In terms of the network simulators, the need for their use in the evaluation of the proposed content distribution strategies in a large-scale manner was clear. Thus, and given the existence of a dataset with all the information needed to input mobility in the emulation process, two new emulators are used as described in Chapter 5. This first one is a Matlab emulator developed to easily design and evaluate content distribution strategies. The other one is specific for developing and evaluating new features and services in the Helix software in a large-scale manner. Thus, this is used to implement and test the proposed content distribution strategies as described in Chapters 5 and 6. Both emulators used mobility information collected in two 24-hour periods (as previously mentioned) as input.

Delay Tolerant Networks Implementations

Several DTN implementations were presented and discussed in this section. After analyzing the widespread DTN implementations, it was concluded that they are not targeted for vehicular environments. Thus, a new DTN implementation developed by IT and Veniam® was presented as a solution for a vehicular network capable of using delay-tolerant mechanisms in order to exchange data among nodes.

Regarding the implementation of several content distribution strategies in a vehicular network, the Helix software will be used in this work as the basis DTN implementation where a new service will be implemented. This service is the content distribution of non-urgent data from remote servers to the mobile nodes using the fixed infrastructure and the vehicular network itself to carry and forward the information. In order to do so, several improvements and modifications are performed in the Helix software as described in Chapter 5.

Chapter 4

Content Distribution Schemes

4.1 Chapter Description

After the description of the fundamental concepts addressed in this Dissertation and along with a description of related work performed in these research areas, it is important to clarify what is the proposed solution in order to achieve an efficient content distribution process. Thus, in this chapter the problem to be addressed is explained along with several strategies to distribute content, whilst maximizing its delivery and consuming the fewest network resources possible. These strategies are defined in accordance with the previous revision of literature.

This chapter is organized as follows:

- *section 4.2 - Problem Statement*: describes the main problem that this work aims to address, which is the design of several content distribution schemes to be implemented in a vehicular network.
- *section 4.3 - Strategies to Stateless Choose Information*: describes how the selection of data to be broadcast by a vehicle is made and presents four different approaches to perform that decision.
- *section 4.4 - Strategies to Disseminate Information*: describes a set of strategies and techniques used to establish a balance between high delivery rates and low medium congestion.
- *section 4.5 - Chapter Considerations*: depicts the conclusions and the summary of the full chapter.

4.2 Problem Statement

A cooperation between the Universities of Aveiro and Porto and a spin-off of both, Veniam, lead to the implementation of the largest – on a global level – vehicular communication platform. This platform comprises more than 600 vehicles (taxis and buses) and fixed stations and is located in Oporto city. This platform has been mainly used for two kinds of services: (i) Internet access for the users of Sociedade de Transportes Coletivos do Porto (STCP) buses and (ii) transportation of sensory information from its source to the core of the network where it can be analyzed.

So, there is a need for the introduction of new kinds of services in order to increase the array of applications and functionalities of this platform. It is within this context that this dissertation studies and implements strategies that support the dissemination of non-urgent information to be used on this vehicular platform.

As previously mentioned (see Chapter 2), vehicular networks have certain particularities when compared with other types of networks due to the constant mobility of the nodes that comprise the network and their wide geographical dispersion. These challenges lead to an intermittent connectivity along with frequent network disruption, hindering the reliable communication and exchange of data among nodes. In order to overcome these challenges, and bearing in mind the transportation of non-urgent information, a DTN software called Helix was specially created to be implemented in the FutureCities project's vehicular network. The implementation of this software widens the array of applications and services that can be provided by the network. These services and applications should be focused on the transfer of non-urgent information such as sensory data, advertisement contents, videos (non-real-time), or tourism-related information.

Even though Helix was fully functional, in this initial phase it was mostly focused on the basic structure of an architecture that supported delay-tolerant communications in a vehicular environment. The work that has been developed up to today only allows for a limited array of services that are mainly focused on the transmission of sensory information to the fixed infrastructure using vehicles as data mules. However, the information is not carried in a multi-hop manner. As such, the information flow starts at the sensor that transfers information to an OBU that will transport it until it finds an RSU. As such, it is possible to conclude that at this time Helix does not have a reliable service implemented for downloading existing information on the fixed infrastructure and transporting it by multi-hop (through several OBUs) on a vehicular network. It is only able to collect data from passing vehicles directly to the network fixed infrastructure.

These constraints are mainly caused by the early stage of development of the Helix routing module. This module is responsible for deciding which packets should be sent or stored, to whom the data is to be transferred, and in which conditions the information should be sent, among other responsibilities. This module is an important part for the introduction of any kind of new service regarding the spreading or sharing of information between the network nodes, such as the content distribution service. As such, the optimization and evolution of the routing module is imperative to create the necessary conditions for the introduction of these services. Moreover, other modifications and implementations must be performed in order to introduce those kinds of services.

As already shown in Chapter 1, Figure 4.1 represents the envisioned scenario of this work. This figure illustrates a dissemination process in which the content is located on remote content servers located in the fixed infrastructure, and must be downloaded by all of the network vehicles. The process of carrying, storing, and forwarding the data packets is the responsibility of the vehicles which spread it along the network using broadcast communication in order to reach the highest delivery rate as possible. The permanent availability of the content is assured by the RSUs which are directly connected to those remote servers which have the content under dissemination stored.

A content dissemination strategy aims to achieve the highest delivery rate as possible, reaching a large number of vehicles, as well as minimizing network resources consumption during the period in which the content is being disseminated. Therefore, a strategy to implement this kind of service must be carefully designed and implemented in order to achieve

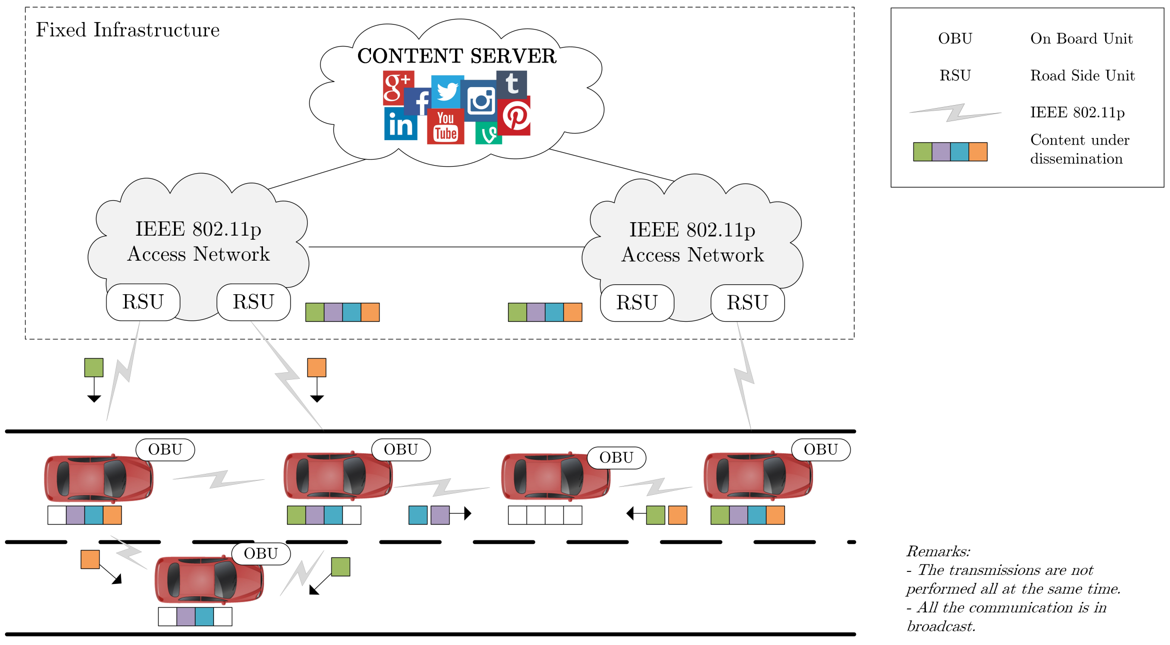

Fixed Infrastructure
CONTENT SERVER
IEEE 802.11p Access Network
IEEE 802.11p Access Network
RSU
RSU
RSU
RSU
OBU
OBU
OBU
OBU
OBU
OBU
On Board Unit
RSU
Road Side Unit
IEEE 802.11p
Content under dissemination
Remarks:
- The transmissions are not performed all at the same time.
- All the communication is in broadcast.

Figure 4.1: Content distribution to-be scenario (illustrative)

such objectives. These goals are addressed when the sender nodes (fixed and mobile) are performing the routing decision. The sender nodes must be able to decide the information to be broadcasted, to what kind of nodes, the time at which it should be sent, the goal of such content, along any other important parameter or characteristic associated with the content under dissemination and the network. Thus, any strategy that aims to implement a content distribution service must address all the previous goals and key factor decisions in order to be correctly deployed.

Within this context, this dissertation aims to deploy a non-urgent content dissemination service (e.g. advertisements, videos, and tourism-related information) distributing contents from remote servers to the network vehicles using delay-tolerant communications. As such, this study focused on two issues: (i) which is the strategy to use in order to select the packets to be sent by broadcast – main focus of the study – and (ii) how to minimize the impact of broadcasting on network congestion and take advantage of mobility profiles in order to optimize delivery.

4.3 Strategies to Stateless Choose Information

This section discusses and describes how the selection of data to be broadcast by a vehicle is made, presenting a set of suitable approaches in order to select the right data to be broadcast.

As previously discussed in section 3.3, a key factor to achieve a high delivery rate is directly related with the process of selecting the right information to be sent. The sender node, and owner of at least a piece of the content under dissemination, must evaluate which is the best packet to be broadcast to its vicinity. However, this is not an easy decision since the sender node needs to know what packets should be sent, when and if it is necessary to send data. When the vehicular network is used to transport and spread the information in broadcast, this factor presents an even greater relevance, since the selection must take into account what is important for the majority of the sender node's vicinity and not only for a specific neighbor node.

Regarding the previous selection and decision challenge, and as discussed in section 3.3, several strategies could be implemented. Some of them can be completely stateless, not introducing additional information in the network in order to select the packets. On the other hand, other strategies resort to additional control data to perform this decision. Typically, this control data tends to be in the form of additional fields in the data packet header providing metadata to help in the sending decision. In a more costly way, additional control packets can be used to spread meta information through the network. As an example, these kinds of packets can carry information about the content of a node's storage, or spread a request for a specific packet or content.

In order to overcome those challenges and as accurately as possible select the data to be broadcast, four strategies have been proposed:

1. Random.
2. Least Number of Hops First (LNHF).
3. LRBF.
4. LRGF.

These strategies are designed assuming that information to be disseminated is stored on an Internet server and is delivered to the VANETs through RSUs, which in turn will be sent to OBUs (vehicles), and then OBUs spread the content through the network.

4.3.1 Random

The first proposed strategy does not require any knowledge about how the content has been disseminated along the network. In order to perform the forwarding decision, the sender node does not know anything about the storage content of its neighbors, or even about its own stored packets. The decision is purely random, which means that the sender node randomly selects packets from its storage and forwards them in broadcast. If any other additional technique or restriction is applied, the sending of information is opportunistic, whereby every time that a node contacts with other nodes it sends randomly selected packets from its storage.

Figure 4.2 illustrates an example of this strategy. The vehicle S randomly selects a packet from its storage to send in broadcast to its vicinity composed by vehicles A, B, and C. Since it only has the packets number 1, 2, 3, and 5, it broadcasts these packets in a random order until there is at least one valid contact. As illustrated, the sender vehicle does not need to know any information about its neighbors' storage content in order to select the packets to be sent.

One of the problems regarding the lack of knowledge about neighbors' storage content is the sending of packets that they already have. On the other hand, this strategy does not introduce any overhead to perform the forwarding decision since it does not use any additional information to perform the routing decision.

The main goal of this strategy is to study the behavior of an approach where no additional information is used. As such, all the decisions are performed according to the already existent data. Through this evaluation it will be possible to establish this strategy as a baseline of comparison with the enhanced strategies further proposed.

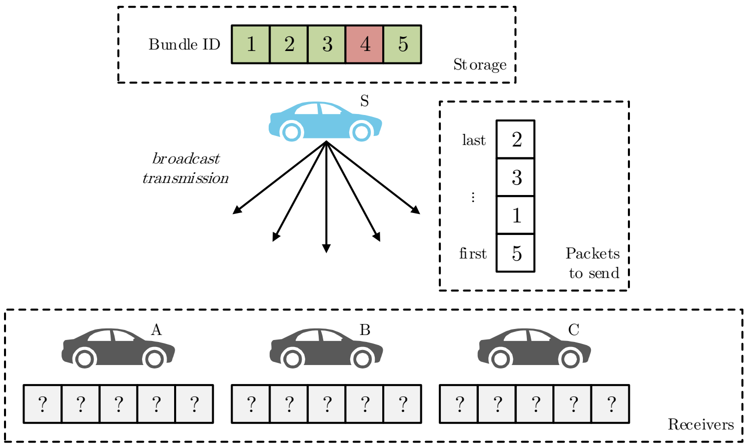


Figure 4.2: Forwarding decision of random strategy - example

4.3.2 Least Number of Hops First (LNHF)

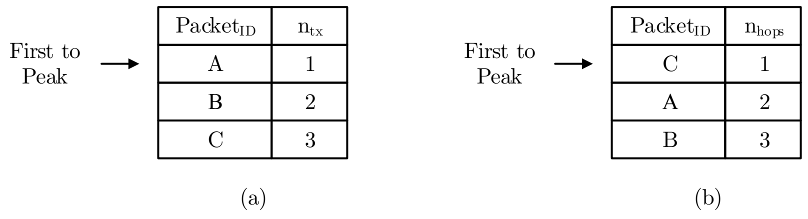
The previous strategy does not require any type of intelligence to perform the forwarding decision. Thus, the next step was to introduce a strategy based on one of two specific characteristics of a packet: (i) number of transmissions, and (ii) number of hops. Moreover, the forwarding decision is performed in a different way according to the type of sender node (OBU or RSU).

The SCF mechanism is the basic concept of spreading a non-urgent content in the network using vehicles as data mules. When a node receives a useful data packet (it is from the content under dissemination and the receiver node does not have it yet), it stores it and carries this data packet until it contacts with other vehicles. When this happens, a copy of the packet is created and sent (in broadcast) to the node's vicinity. When the neighbors receive the packet, they update its internal information about the number of hops this packet already has. Therefore, the content dissemination procedure is based on a multi-hop transfer of data among the network vehicles, whereby the number of hops of a packet is directly related to the number of nodes which already have a specific packet. This parameter can be easily used by an OBU to decide which packet should be sent, without introducing a high network overhead (a couple of extra bytes in the data packet header should be enough to implement it).

Another important factor is the number of transmissions of a packet. As mentioned before, in a content dissemination process the sender node forwards a copy of its locally stored data, whereby it can count how many times it creates a copy of the same data packet to be forwarded to its neighbors. The higher the number of copies (or transmissions), the higher is the probability of this packet already being stored by other nodes when compared to other packets which have a lower number of copies. Regarding this consideration, this parameter can be used as a metric in a content dissemination strategy in order to perform the decision of which packet should be sent. Considering the specific case of the vehicular network fixed infrastructure, the RSUs, this could be an important factor since the other mentioned parameter (number of hops) cannot be used because the number of hops in an RSU is always equal to zero (considering that the connection between the Internet server and the RSU does not increase the number of hops).

As mentioned before, regarding the addition of these two parameters in the data packet header (number of hops and number of transmissions), a new strategy is proposed that selects the packet to be sent according to the number of hops (if the sender node is an OBU), or according to the number of transmissions (if the sender node is an RSU).

Regarding the conceptual implementation of this strategy, it is clear that the sender nodes must have internal structures to allow a fast selection of the packet to be sent. This is achieved through the mapping between the packet ID and one of the mentioned parameters (number of hops or number of transmissions). On the other hand, these auxiliary structures must be different according to the node type since the selection decision is different between an RSU and an OBU. An example of these internal structures is illustrated in Figure 4.3 where it is clear that the first packet selected by an RSU must be the packet A since it has a lower number of transmission. On the other hand, the OBU would select the packet C because it has a lower number of hops.



$\text{Packet}_{\text{ID}}$	n_{tx}
A	1
B	2
C	3

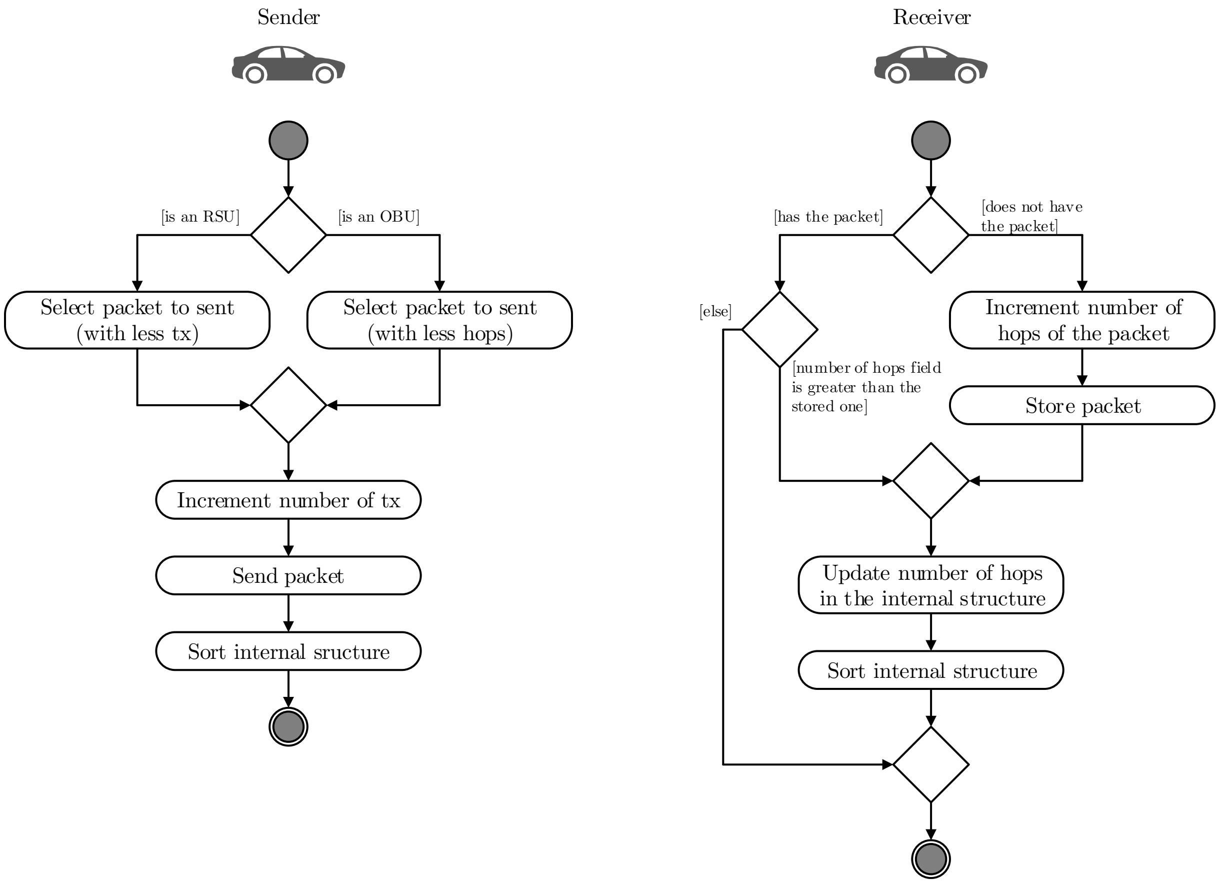
$\text{Packet}_{\text{ID}}$	n_{hops}
C	1
A	2
B	3

Figure 4.3: Internal stractures of LNHF strategy in an (a) RSU and (b) an OBU - conceptual

Figure 4.4 summarizes the previously described procedures of transmission and reception when this strategy is applied. Thus, when a node has at least one valid neighbor (this is an opportunistic strategy), if it is an OBU it will check the internal structure which maps the packet ID according to its number of hops, and selects the first one. On the other hand, if it is an RSU, the packet with less transmissions is selected. Once the packet is evaluated, its number of transmissions is incremented and it is sent in broadcast. Since one of the sorting factors changed, the internal structure must be re-sorted before the end of the process. From the receiver's point of view, if the node already has this specific packet, it checks the number of hops recorded for this packet in its internal structure. If the received number of hops is greater than the one currently stored, the higher value is assumed and the internal structure is updated. However, if the receiver node does not have the packet, it increments the number of hops field in the actual packet along with an update in its internal structures. Once again, given that one of the sorting factors changed, the internal structure must be rearranged.

Figure 4.5 illustrates an example of the forwarding decision of this strategy. The sender node does not have any information about its neighbors' storage content. Therefore, if the sender node is an RSU (Figure 4.5-(a)), the packets are selected according to the number of previous transmissions, and the packets with a lower number of transmissions are forwarded first, whereby packet number 1 is sent first. On the other hand, if the sender is an OBU, the criteria is the number of previous hops. Thus, as illustrated in Figure 4.5-(b), the first packet being sent by node S is the number 5, since it is the one with less number of hops, followed by 1, 3, and 2. When a node receives a certain packet that already was stored, it updates the number of hops if the received packet has a higher number of hops than the currently stored one.

Contrarily to the first one, this strategy requires the addition of two new fields to store the number of hops and number of transmissions. Due to this, an increase in the network



Sender
[is an RSU]
[is an OBU]
Select packet to sent (with less tx)
Select packet to sent (with less hops)
Increment number of tx
Send packet
Sort internal sructure
Receiver
[has the packet]
[does not have the packet]
[else]
[number of hops field is greater than the stored one]
Increment number of hops of the packet
Store packet
Update number of hops in the internal structure
Sort internal structure

Figure 4.4: Procedures for sending and receiving of data packets in the LNHF strategy - conceptual

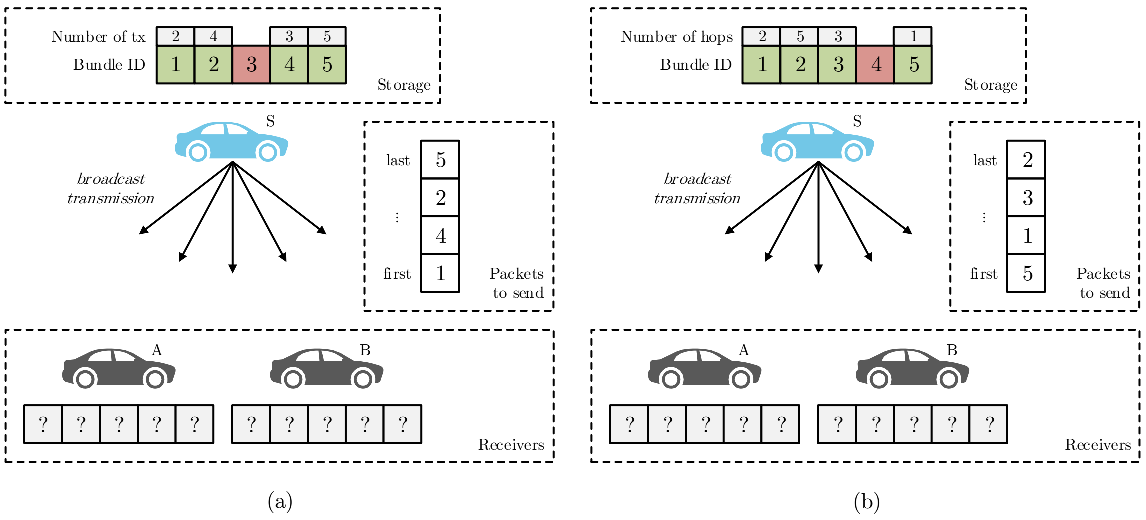


Figure 4.5: Forwarding decision of LNHF strategy in a (a) RSU and (b) an OBU - example

overhead is expected.

4.3.3 Local Rarest Bundle First (LRBF)

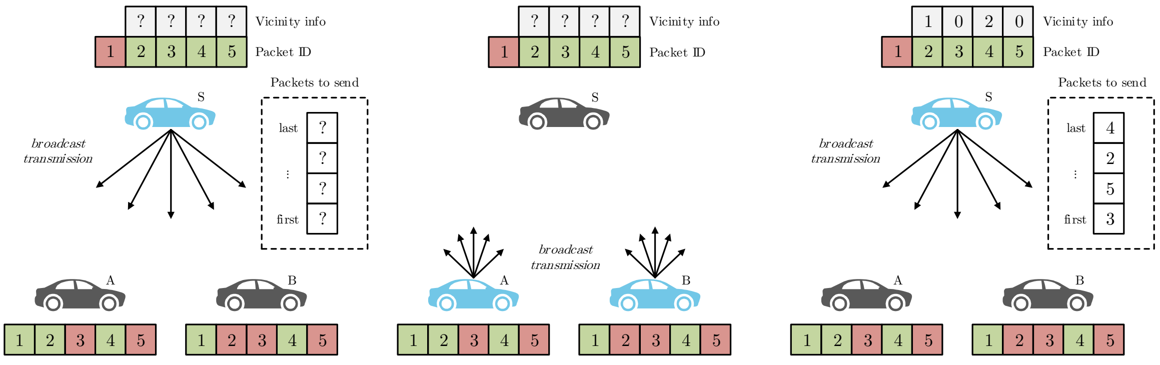
As mentioned in the beginning of this chapter, a major challenge associated with a content dissemination strategy is the decision of selecting which packet should be sent in order to achieve the highest delivery rate possible. The first two proposed strategies aim to perform this selection using as minimum control data as possible in order to not increase the network overhead. However, in this strategy a quite different approach is proposed, which focuses on the sender node's vicinity storage content.

The ideal situation would be the one where the sender node has exact knowledge of the storage content of its vicinity. Moreover, as the communication is broadcast, the ideal situation would be one where the sender node exactly knows which is the most lacking content in its vicinity. The implementation of a strategy that covers and achieves this ideal situation tends to be very costly in terms of network resource consumption and can increase the network overhead since the introduction of new control packets to spread the state information through the network would be necessary. Although the implementation could lead to the previous drawbacks, the potential advantages introduced by this kind of strategy justifies its design, development, and evaluation for the challenge that this work is proposing to overcome.

The design of this strategy is based on a P2P communication which uses a *local rarest packet* first policy similar to the *rarest piece* first download policy in torrent schemes: the least available packet (measured in terms of the number of neighboring nodes having the packet) is selected. Thus, each sender node needs to be aware of its vicinity storage content in order to select the least available packet. In order to perform this decision several modifications and new features need to be added. The following topics aim to address these new features and how these could be implemented in a real vehicular network.

The conceptual design of this strategy is illustrated in Figure 4.6. This figure aims to represent the As-Is and the To-Be situations. The first one represents a situation where the sender node does not know which packets should be sent since it does not have any content distribution strategy implemented. As an evolution of this scenario and a way to implement the proposed strategy, the nodes A and B can periodically send a control packet advertising their storage content. In an ideal situation, the sender node knows the exact content of its neighbors' storage. Regarding the example illustrated, vehicle A advertises that within its storage it has the packets 1, 2 and 4 from a specific content, and node B advertises that it has packets 1 and 4. Through these advertisements, in the to-be scenario, the initial sender node (S) gets to know which are the packets that it needs to broadcast in order to send the most useful information for its neighbors. Thus, given that the most lacking packets are the packets 3 and 5, these are first ones to be broadcast by node S, followed by packets 2 and 4, respectively. In order to perform this sending decision as fast as possible, the node needs to have an auxiliary internal structure to map the stored packets in node S with the number of neighbors that have those packets. As a footnote, and given the previously discussed LNHF strategy, when two packets are equally lacking in the sender node's vicinity, the tiebreaking criteria could be the number of hops (in an OBU), or the number of packets transmission (in an RSU).

From the conceptual point of view illustrated in Figure 4.6, it is possible to understand the necessity to create a set of auxiliary internal structures. The sender node must have an internal structure (such as a table) where it maps its own packets with the information



(a) (b) (c)

Figure 4.6: LRBF strategy (a) As-Is, (b) intermediate, and (c) To-Be scenarios

collected from the network advertisements. This table or list must be sorted in order to facilitate the sending decision, since the sender node only looks to the first position of such table, picks the packet and broadcasts it. However, the information contained in this table cannot be perpetual, whereby it must have a validation period, otherwise the sender node could be using dated information to perform the forwarding decision. The decision is directly related to the need to send the most lacking packets. Moreover, it has to be a compromise between the refreshing time of those structures and the sending of packets, to maximize the number of correct decisions and to keep the node's resources consumption as low as possible.

Once the conceptual point of view of this strategy is explained and structured, it is important to address the following question: how can the internal structures be updated in order to perform the forwarding decision in a high mobility and mesh network? As suggested by Figure 4.6-(b), the answer could be the use of additional control packets which advertise the storage content of each node. These packets must have a set of fields in order to identify the node, its contents, and the pieces of each content that it has (see an example in Figure 4.7). Although this kind of information is essential, it is important to keep in mind that it has a cost, the bigger the content (or contents) under dissemination, the bigger the advertisement packet, leading to a potential network overhead increase. Another relevant factor is the periodicity of these advertisement packets. The higher the frequency, the more accurate the forwarding decision tends to be and the higher the delivery rate, since the sender node has more accurate information. However, this pace cannot be too fast in order to keep the network overhead in moderate value. In order to spread the state information through the maximum number of vehicles as possible, these messages are broadcast by each vehicle. Moreover, the internal structures can also be updated through the analysis of the received packets. Thus, when a vehicle receives a data packet from another node, it gets to know that the sender node already has a certain packet, whereby it can upload its internal structures to reflect this new information.

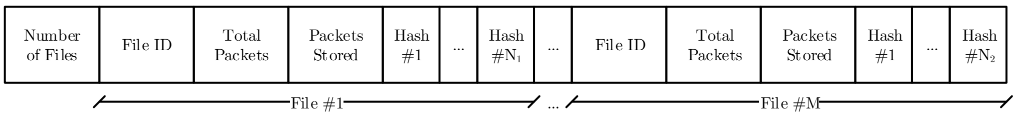


Figure 4.7: LRBF strategy advertisement packet structure - example

Figure 4.8 illustrates the procedure to send an advertisement packet and how a node interprets a received one, when only a single file is under dissemination. Thus, in order to send an advertisement packet, the sender node must first collect information about its own storage in order to gather data about contents under disseminations: file identifier(s), packets stored of each file (number and identifiers), along with other future relevant information. Once this set of information is collected, the advertisement packet can be constructed and sent in broadcast to the node's vicinity. When a node receives an advertisement packet, first of all it needs to identify the source node since this information is crucial to correctly update its internal structures. The advertisement could carry information about several contents under dissemination, whereby the receiver node must identify each one of them and which packets the sender node has of each content. Once this information is collected, the receiver node can update the internal structures responsible for keeping a state of which is the most lacking packet in a node's vicinity. As the sorting criteria of the internal structure responsible for retrieving the first packet to be picked from storage may have been changed, this structure must be re-sorted.

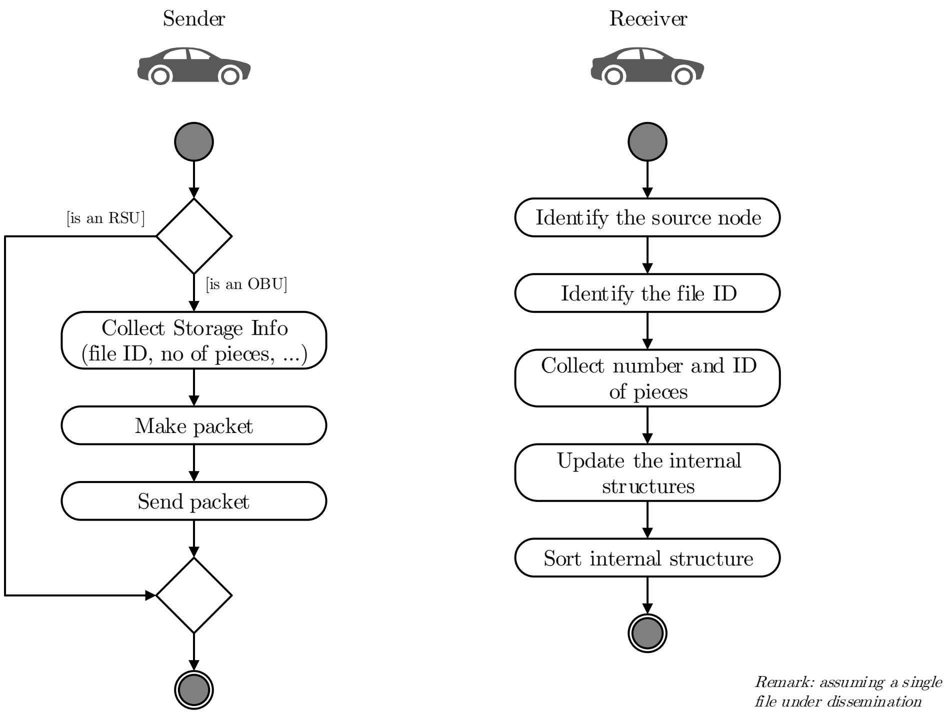

Sender
Receiver
[is an RSU]
[is an OBU]
Collect Storage Info (file ID, no of pieces, ...)
Make packet
Send packet
Identify the source node
Identify the file ID
Collect number and ID of pieces
Update the internal structures
Sort internal structure
Remark: assuming a single file under dissemination

Figure 4.8: Procedures for sending and receiving of advertisement packets in the LRBF strategy - conceptual

Another important procedure is the refreshment of the internal structures. As mentioned before, the auxiliary structures are built using information from advertisement packets and received data packets. This information is directly related to its vicinity, whereby, due to the high network mobility, they are not continuously valid. Thus, a refreshment procedure must occur to remove the dated information which could lead to inaccurate forwarding decisions. Each entry of information must have a timestamp attached to facilitate the refreshment procedure. As an example, when a node receives an advertisement packet announcing that node X has the packet number 1 of file A, a timestamp must be attached or the previous one must be replaced in the internal structure responsible for counting the number of neighbors that have the same packet. Each entry that exceeds a specified valid time period must be deleted and the internal structures must be re-sorted. This mechanism enables a more accurate and correct forwarding decision.

Figure 4.9 summarizes the previously described procedures of transmission and reception when this strategy is applied. Thus, when a node has at least one valid neighbor, which according to the sender node's internal structures needs at least one packet that the sender node has, the transmission will occur. The sender node will check the internal structure which maps the packet ID according to the most lacking packet, and selects the first one. Once the packet is evaluated, its number of transmissions is incremented and sent in broadcast. Since one of the sorting factors changed, the internal structure must be re-sorted before the end of the process.

From the receiver's point of view, if the node already has this specific packet, the corresponding entry in the internal structure responsible for controlling how many neighbors this specific packet has, is updated. However, if the receiver node does not have the packet, it creates a new entry in the internal structure and stores the packet. Once again, as one of the sorting factors changed, the internal structure must be rearranged.

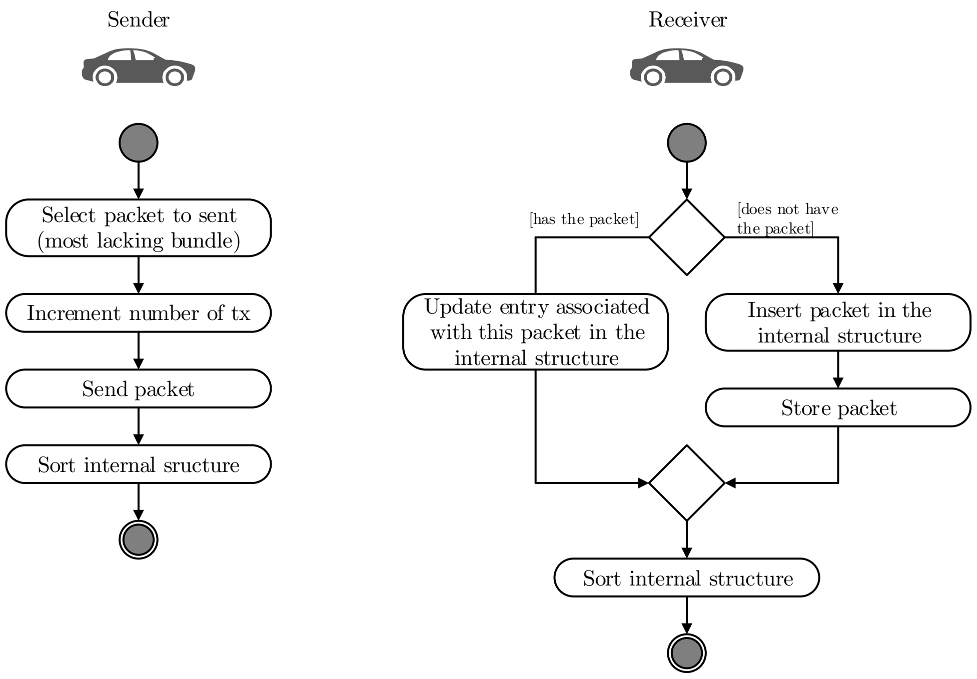

Sender
Receiver
Select packet to sent (most lacking bundle)
Increment number of tx
Send packet
Sort internal sructure
[has the packet]
[does not have the packet]
Update entry associated with this packet in the internal structure
Insert packet in the internal structure
Store packet
Sort internal structure

Figure 4.9: Procedures for sending and receiving of data packets in the LRBF strategy - conceptual

Contrarily to the LNHF strategy, in this approach the transmission of information is not fully-opportunistic since the nodes are not sending data every time a valid contact is established. The LRBF strategy should implement a mechanism that saves resources in order to minimize the number of transmission conducing to lower network congestion. The optimization of the forwarding decision (when should a node send information) is associated with its vicinity. If at least one neighbor needs a packet that the sender node has, it must broadcast this packet. If not, the sender nodes must stop the dissemination process.

Thus, at the beginning of the dissemination process, only RSUs are authorized to send advertisement packets and any node can send data packets. Figure 4.10-(a) illustrates the procedure to enable the broadcast of data packets. Node A, which is able to send advertisement packets (that at the beginning can only be RSUs but after a time, also OBUs), periodically sends these announcing its storage content. When an OBU which does not have any content yet receives this advertisement (node B), it gets to know that there is content under dissemination, whereby it updates the internal structures and starts broadcasting advertisement packets. When node A receives this announcement, it understands that there is at least one neighbor which does not have any data associated with the content under dissemination. Moreover, if the advertisement reports incomplete content and the receiver node has data that is useful to the advertiser (data that it does not have), it spreads useful data packets in broadcast.

The opposite situation is described in Figure 4.10-(b). A node should end the process of dissemination if the current neighbors have completed the download of the content or when it does not have any neighbors. Thus, when one (or both) of these conditions are met, the refreshment procedure occurs in order to clean all the dated information from the internal structure. If, at the end of this process the internal structure is empty, the node ends the dissemination of data packets, stopping the content dissemination procedure.

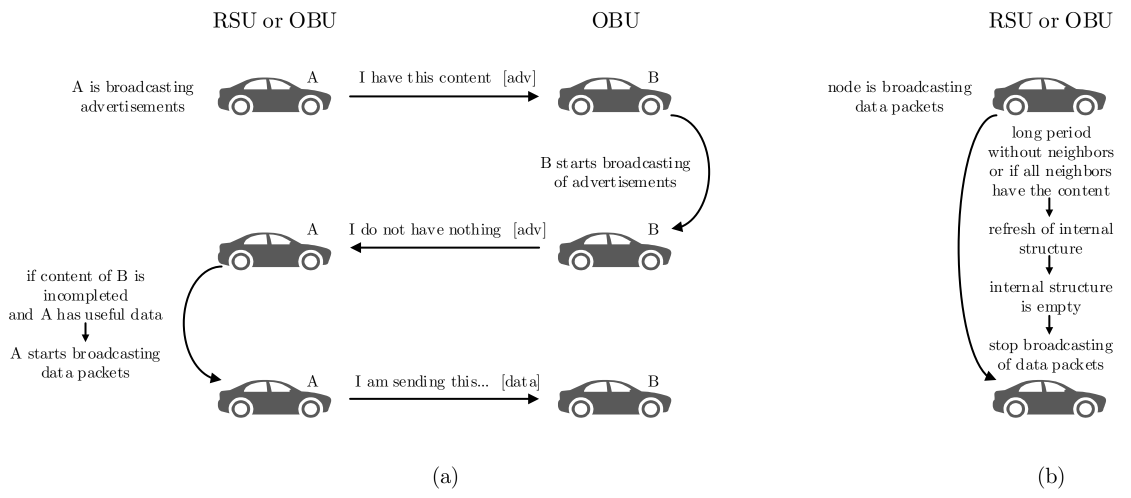

RSU or OBU
OBU
RSU or OBU
A is broadcasting advertisements
A
I have this content [adv]
B
B starts broadcasting of advertisements
A
I do not have nothing [adv]
B
if content of B is incompleted and A has useful data
A starts broadcasting data packets
A
I am sending this... [data]
B
node is broadcasting data packets
long period without neighbors or if all neighbors have the content
refresh of internal structure
internal structure is empty
stop broadcasting of data packets
(a)
(b)

Figure 4.10: (a) start and (b) end of dissemination in the LRBF strategy

Figure 4.11 illustrates an example of a forwarding decision following this strategy assuming that all of the advertisement process has occurred immediately before. The packets number 2 and 3 are lacking in all the neighbors whereby these are the first packets to be sent by node S. After these, packet number 5 is sent since it is lacking in two neighbors (nodes B and C). Finally, packet number 4 is sent since it is lacking in one neighbor (node A). Although packet number 1 is lacking also in one single neighbor (node C), it is not sent since the sender node does not have it.

4.3.4 Local Rarest Generation First (LRGF)

In this subsection another strategy to select which packet should be sent is proposed. This strategy is based on the same principles as the previous one, but it considers the *rarest coded*

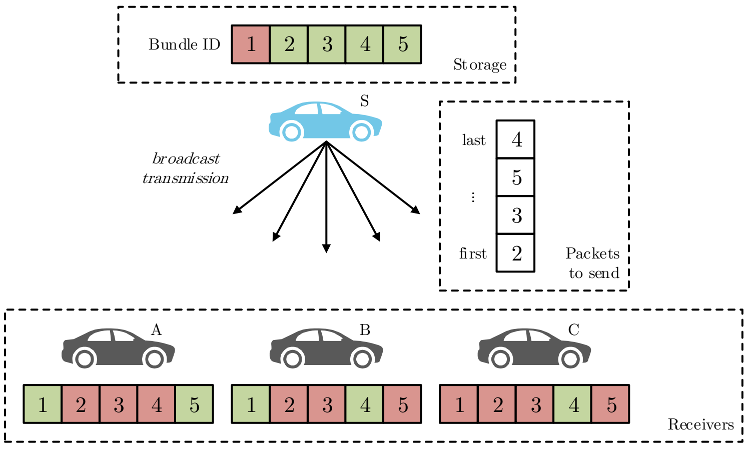


Figure 4.11: Forwarding decision of LRBF strategy - example

generation instead of the *rarest packet* to be sent.

As explained in subsubsection 3.3.2.2, the use of network coding in content distribution schemes brings a set of advantages. This new approach leads to a better usage of the available bandwidth, allowing the maximization of the multicast capacity. These fewer transmissions can also lead to a better energy-efficiency which could be an important factor in mobile networks. Network coding does not need to establish or find the optimal path from source to destination to achieve the required throughput. Considering a many-to-all broadcast network, similar to a content distribution scenario, the flooding of packets could not be the ideal scenario due to the high overhead and high level of rebroadcasting; however, with network coding the rebroadcast rate is reduced from $O(NlogN)$ to $O(N)$ [153].

On the other hand, network coding can lead to a set of drawbacks. Contrarily to the packet routing approach, a completely decoded packet cannot be sent directly to its destination since it is coded in multiple packets and the receiver node needs a set of coded packets in order to decode the sent packet. In network coding, an intermediate node must wait for a number of packets so that they can be combined, increasing the latency of the first packet. Another drawback can be its robustness to lost packets. If one packet is lost, it can affect the decoding of a certain packet; however, this drawback is most of the time overcame due to the redundancy introduced by the coding procedure.

Using the concept of network coding, the original file is divided into frames by the application which are then reorganized into a group of blocks (identified by *blockid*), each of which is a set of *blocksize* adjacent frames. Assuming Random Linear Network Coding (RLNC) [154], a coded packet is a random linear combination of frames which is coded using a vector element in a certain finite field (typically a Galois Field). The encoding vector is stored in the header of a coded packet, along with *blockid* and *blocksize* for the purpose of further decoding at the receivers. A set of coded packets belongs to the same generation if they are associated with the same *blockid* and a generation has *gensize* coded packets. To generate a coded packet of a certain generation, *blocksize* frames of the block associated with that generation are required. Upon receiving a coded packet, every node stores the packet in local memory for further decoding. In order to decode all of the *blocksize* frames belonging to the same block, a node needs to collect more than or an equal number of *blocksize* coded packets belonging to the same generation, and their encoding vectors that are linearly independent of each other.

Figure 4.12 illustrates the previous procedure which is based on CodeCast protocol [68]. The proposed strategy is based on RLNC but without re-codification in intermediary nodes, so an intermediary node can only generate coded packets when it has a complete decoded block or forward coded packets coded by other nodes.

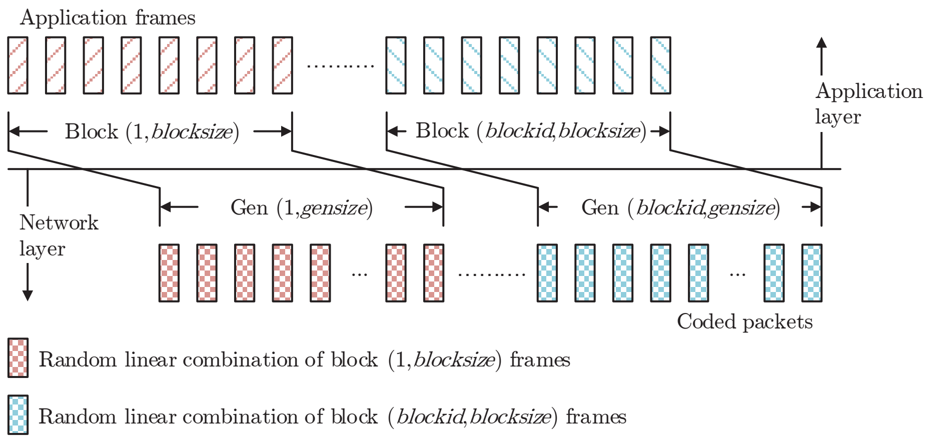

Application frames
Block (1,*blocksize*)
Block (*blockid*,*blocksize*)
Application layer
Network layer
Gen (1,*gensize*)
Gen (*blockid*,*gensize*)
Coded packets
Random linear combination of block (1,*blocksize*) frames
Random linear combination of block (*blockid*,*blocksize*) frames

Figure 4.12: Relationship between application frames, blocks, and coded packets, based on [68]

In this case, it is considered that the number of coded packets from the same generation available to be forward by the sender node is recorded in a field called *rank*, in the header of the forward coded packet. A coded packet with a rank smaller than *blocksize* indicates that the vehicle A is in need of more coded packets of that generation. On the other hand, if the *rank* is equal to *blocksize* it means that the vehicle A already has decoded the associated block, which means that there is no need to send coded packets from this generation. In response to a rank lower than *blocksize*, vehicle B transmits more coded packets from the associated generation to help vehicle A in the collection of additional coded packets of the most lacking generation. With the *rank* information, it is not important to know the exact packet lacking in a node, but only the amount of coded packets of a specific generation that need to be sent in order to decode a block. This mechanism tends to decrease the network overhead since the nodes do not exchange their storage content (such as in the LRBF strategy), and only share information about the content generations, which is always smaller (assuming *blocksize* greater than zero and equal sized packet fields).

The proposed strategy aims to implement a method to select which coded packets should be sent. When a node collects *blocksize* coded packets, it will automatically generate the additional *gensize* minus *blocksize* coded packets, leading to a higher network redundancy. Once these coded packets are generated, the LRGF will perform its actions, choosing the most suitable coded packet to be sent. Moreover, during the implementation it will be also assumed that there is no re-codification in the intermediary nodes, whereby a coded packet can only be coded once, and cannot be combined with other coded packets.

The practical implementation (not the conceptual view) of this strategy is not too different from the LRBF strategy. An overview of the conceptual design of this strategy is illustrated in Figure 4.13, which represents the current situation to select a packet to send (a), the future moment of the forwarding decision (c), using (b) as an intermediate step to achieve the proposed packet selection strategy. Similarly to the previous strategy, in the first one the sender node does not know which coded packet should be forwarded since no forwarding decision is implemented. In order to implement the proposed strategy, nodes A and B should periodically disseminate an advertisement packet with their ranking. As mentioned before, the ranking is a metric of how many coded packets, associated with a specific block/generation, a node has within its storage. Through these advertisements the sender node S can evaluate which is the generation of coded packets that is most lacking in its vicinity and randomly choose coded packets from this generation to be broadcasted. In the Figure 4.13 example, vehicle A advertises that it has a rank of 2 from generations 1, 2, and 3, and a rank of zero regarding the generations 3 and 5. On the other hand, vehicle B has a rank of 0 from generations 2, 3, and 5, and a rank of 2 from generations 1 and 4. Thus, using the advertised data, sender node S selects and broadcasts the most useful information to its vicinity. Thus, the most lacking generations are the ones identified by numbers 3 and 5, followed by generation 2. Similarly to the LRBF, in order to perform this forwarding decision as fast as possible, the node should have an auxiliary internal structure to map the generation which the sender node already has (completed or not completed) with the ranking criteria collected from its vicinity. Regarding the previously discussed LNHF strategy, when two generations are equally lacking in the sender node's vicinity, the tiebreaking criteria could be the number of transmissions associated with these generations (being equal for OBUs and RSUs).

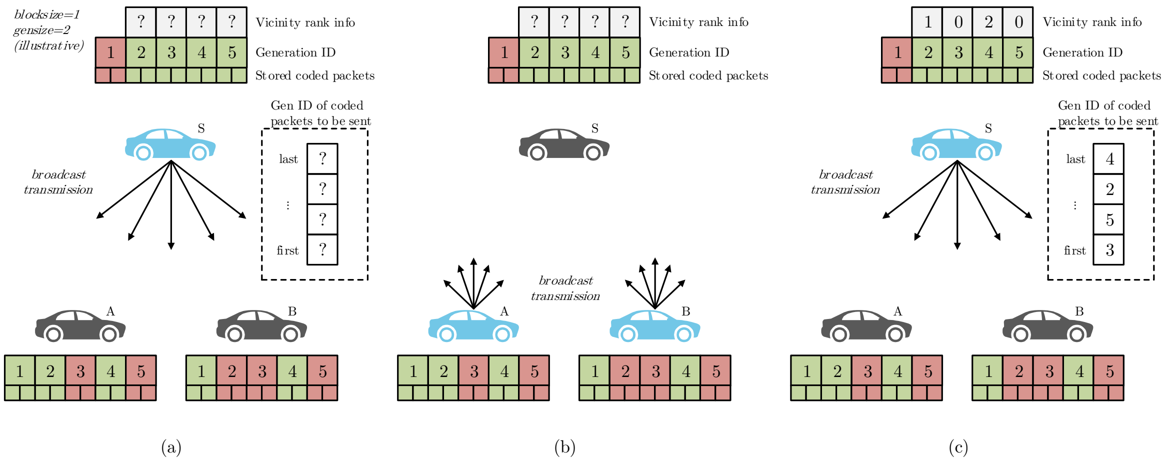


Figure 4.13: LRGF strategy (a) As-Is, (b) intermediate, and (c) To-Be scenarios

Regarding the conceptual point of view previously illustrated in Figure 4.13, the necessity to create a set of support structures is clear. Similarly to the LRBF strategy, the sender node must have an internal structure where the collected ranking information is mapped with the generations stored (completed or not completed). This map must be sorted in order to optimize the forwarding decision, since through this approach the sender node immediately identifies the generation to be spread (the most lacking generation). Moreover, two situations can be considered: (i) this structure only includes completed generations (with *gensize* coded packets available to be sent), or (ii) uncompleted generations (with less than *gensize* coded packets available). An example of both situations will be further explained in this section. However, the information of this map cannot be perpetual, whereby it must have a maximum validity time, to prevent the forwarding of data based on dated control information.

Once again, and similarly to the LRBF strategy, the same approach is used to update those support structures. Additional control packets are periodically advertised by the network nodes, spreading information about their storage content. These messages are broadcasted by each vehicle to disseminate the state information through the maximum number of vehicles as possible. These packets must have a set of fields in order to identify the node, which contents/files it has, and which is the ranking of each generation of these contents (see an example in Figure 4.14). Moreover, other generic fields can be added such as the *blocksize* and *genesize* specified for each content. Once again, although the disseminated information is essential for the forwarding decision, it has a cost. The bigger the content (or contents) under dissemination and the lower the *blocksize* (more blocks), the bigger the advertisement packet will be, leading to a potential network overhead increase. The periodicity of these advertisement packets is also an important factor. A compromise must be established since the higher the periodicity, the more accurate the forwarding decision tends to be (leading to a higher delivery rate). However, if this parameter is too high, the network overhead can increase excessively.

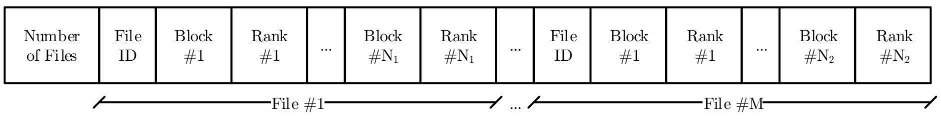


Figure 4.14: LRGF strategy advertisement packet structure - example

Figure 4.15 illustrates the processes taken to send an advertisement packet and to handle a received one, when only a single file is under dissemination. The process to send an advertisement packet starts with the collection of storage information such as the file ID(s) under dissemination, and for each one of them it can collect the associated *blocksize* and *gensize*, along with the ranking of each block. As mentioned before, this ranking is limited to *blocksize* and should retrieve the number of coded packets associated with a specific block/generation which the sender node has stored. Once all of this information is collected, the advertisement packet is made and sent in broadcast to the node's vicinity. On the other hand, when a node receives an advertisement packet, it identifies the node which has sent the packet. For each reported content, it collects the file identifier and, if the receiver node already has the reported block, it also collects the rank associated with the block. If more than one file is reported in the advertisement packet, the last procedure is repeated for all of them. Once this information is collected, the receiver node can update the internal structures responsible for keeping a state of which is the most lacking generation in node's vicinity. With the collection of new information and subsequent update of the internal structures, the sorting criteria of the structure responsible for signaling the most lacking generation may have been changed, whereby this structure must be re-sorted.

As mentioned for the LRBF, the refreshment of the internal structures responsible for identifying the next packet or, in the case of the LRGF strategy, the block/generation to which the next coded packet belongs, is a crucial procedure. This structure is created and updated using the advertisement packets periodically broadcasted by each node. As highlighted before, a vehicular network is characterized by intense node mobility, implying a short valid time for the broadcasted information. Thus, in order to overcome inaccurate forwarding decisions, a refreshment procedure must occur. Similarly to the LRBF, each entry of the mapping table must have a timestamp attached to facilitate the refreshment procedure. Each one of these entries (one for each generation) should be a list of three parameters: neighbor identifier, rank, and timestamp of when this information was collected. Each entry that exceeds a specified valid time, must be deleted, and the internal structures re-sorted, in order to ensure a more

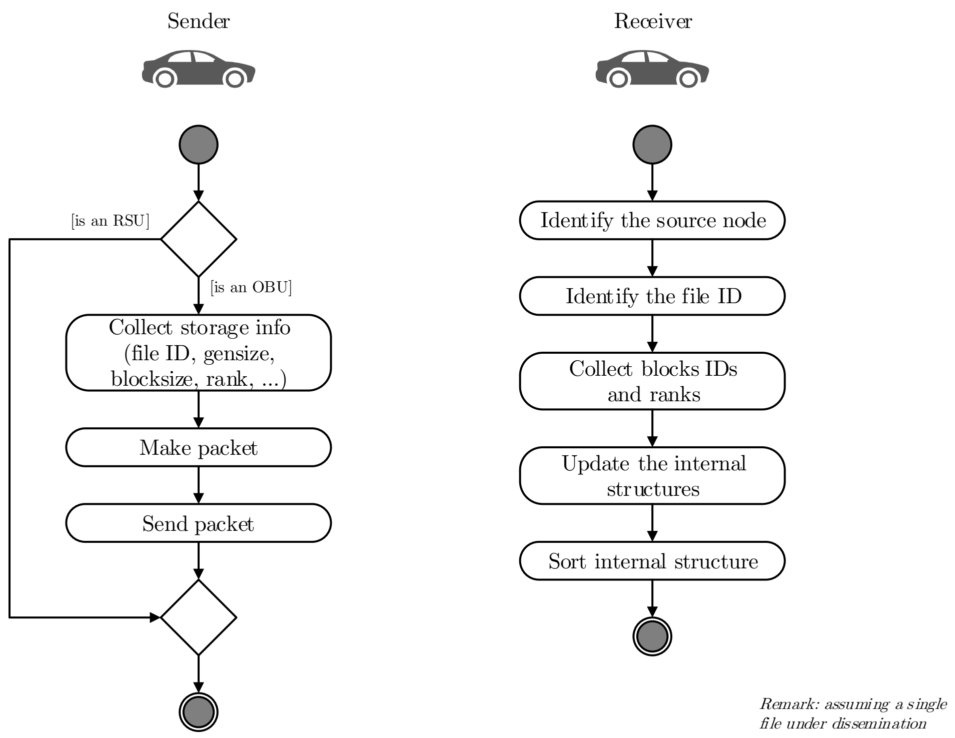
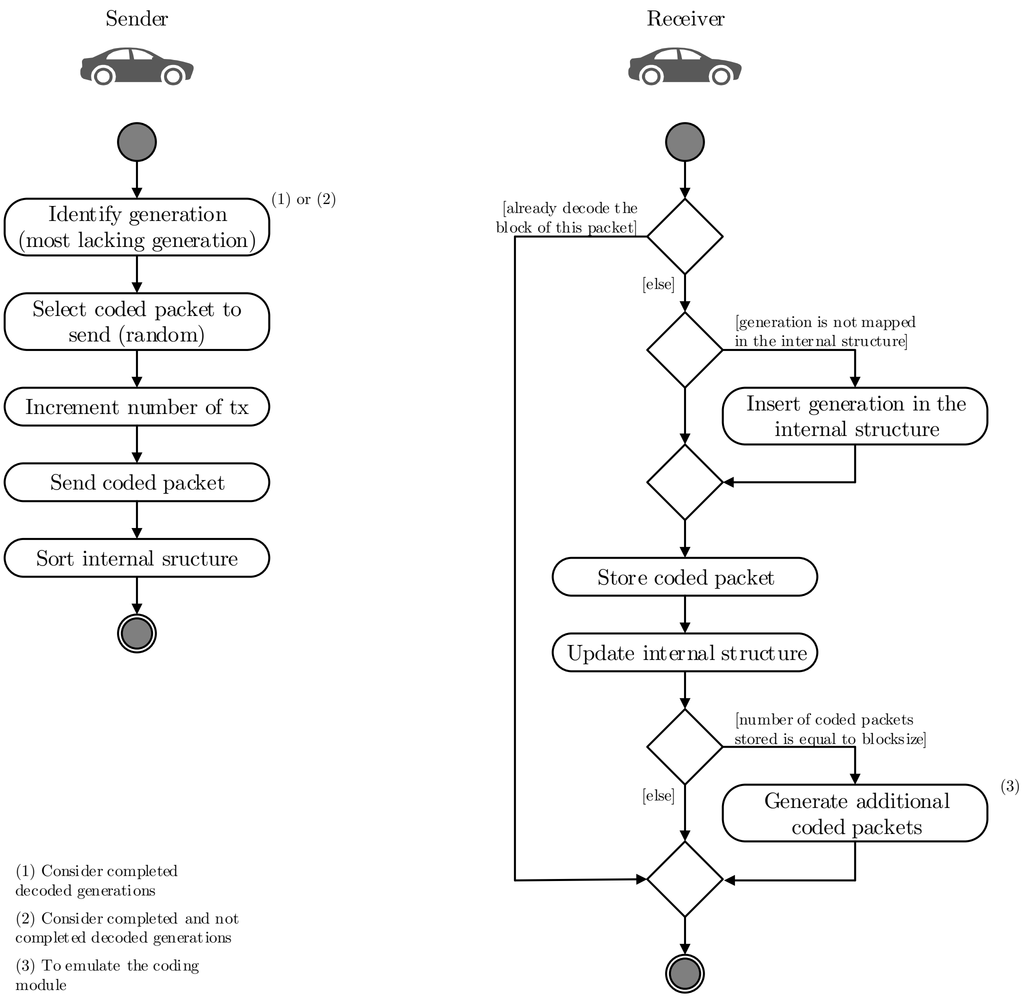

Sender
Receiver
[is an RSU]
[is an OBU]
Collect storage info (file ID, gensize, blocksize, rank, ...)
Make packet
Send packet
Identify the source node
Identify the file ID
Collect blocks IDs and ranks
Update the internal structures
Sort internal structure
Remark: assuming a single file under dissemination

Figure 4.15: Procedures for sending and receiving of advertisement packets in the LRGF strategy - conceptual

accurate and correct forwarding decision.

Figure 4.16 summarizes the procedures of sending and receiving data packets when this strategy is applied. When the sender node has at least one valid neighbor which, according to the internal mapping of the sender node needs at least one coded packet associated with a generation that the sender node has, the transmission will occur. First, the sender node identifies the most lacking generation (through the internal mapping structure), and then randomly selects a coded packet from this generation to be broadcasted. As remarked before, there are two possible situations: (i) the sender node is only able to forward coded packets which belong to a completely decoded block/generation, or (ii) it can forward coded packets from a block/generation which has not been decoded yet. This second situation is not an operation of recoding but only an additional feature of the LRGF strategy. Once the packet is evaluated, its number of transmissions is incremented and sent in broadcast. Since one of the sort factors changed, the internal structure must be re-sorted before the end of the process.

When a node receives a coded data packet, if it has already decoded the associated block, it does not do anything. On the other hand, if there is no entry related to this coded packet (same file ID, block and generation numbers) in the internal structure, a new entry (addressing the received block's generation) must be added to the internal structure. After that, the received coded packet is stored and the internal structure responsible for mapping the node's storage (maps the block/generation with the number of coded packets stored of those blocks/generations) is updated, signaling the existence of a new coded packet. This occurs when a node receives at least *blocksize* coded packets from the same generation. From this moment on, the node has *gensize* coded packets associated to the same block and generation available to broadcast.



Sender
Receiver
Identify generation (most lacking generation)
(1) or (2)
Select coded packet to send (random)
Increment number of tx
Send coded packet
Sort internal sructure
[already decode the block of this packet]
[else]
[generation is not mapped in the internal structure]
Insert generation in the internal structure
Store coded packet
Update internal structure
[number of coded packets stored is equal to blocksize]
[else]
Generate additional coded packets
(3)
(1) Consider completed decoded generations
(2) Consider completed and not completed decoded generations
(3) To emulate the coding module

Figure 4.16: Procedures for sending and receiving data of packets in the LRGF strategy - conceptual

Figure 4.17 illustrates the start and the end of the dissemination procedure for the LRGF strategy. The start and the end of the dissemination process are defined in the exact same way as in the LRBF strategy. The only difference is the fact that, in the LRBF strategy, the nodes exchange packets between them, and in the LRGF there are coded packets being exchanged among the network nodes.

As mentioned before, even though the node always sends the rarest generation, the forwarding decision can be performed in one of two ways, which are represented in the following examples.

The first one is illustrated in Figure 4.18. In this approach the sender node can only send coded packets from a generation associated with an already decoded block. In the example illustrated in Figure 4.18, the most lacking generation is the generation number 1 with a total of 1 coded packet among all the neighboring nodes (A, B, and C). Thus, the selected generation to send coded packets is number one. However, the sender node did not decode the block associated with this generation yet. Due to this, coded packets of generation number

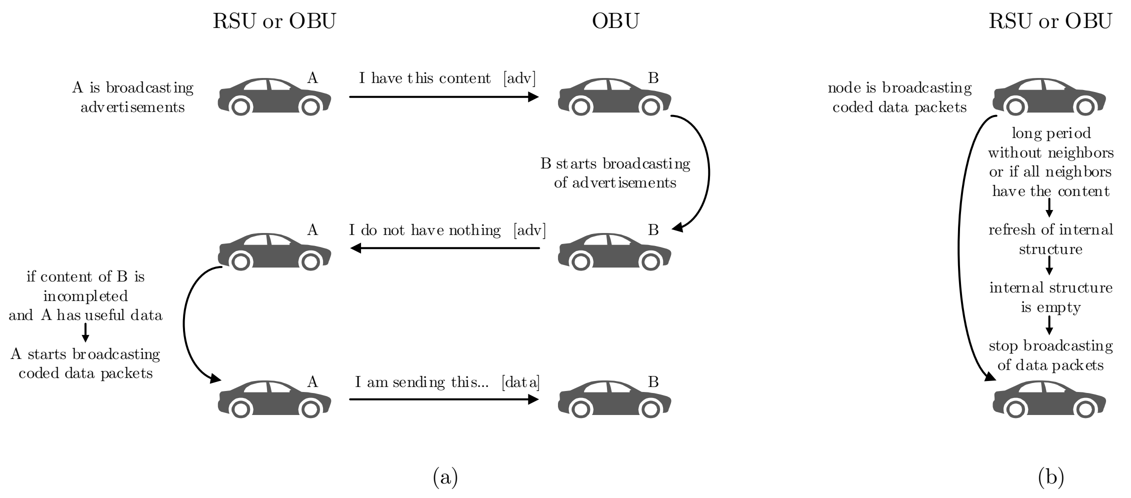

RSU or OBU
OBU
RSU or OBU
A is broadcasting advertisements
A
I have this content [adv]
B
B starts broadcasting of advertisements
A
I do not have nothing [adv]
B
if content of B is incompleted and A has useful data
A starts broadcasting coded data packets
A
I am sending this... [data]
B
node is broadcasting coded data packets
long period without neighbors or if all neighbors have the content
refresh of internal structure
internal structure is empty
stop broadcasting of data packets
(a)
(b)

Figure 4.17: (a) start and (b) end of dissemination in LRGF strategy

three will be sent, since it is the second most lacking generation and the sender node already decoded the block associated with it. Once the generation to send is identified, the coded packets are randomly selected among them.

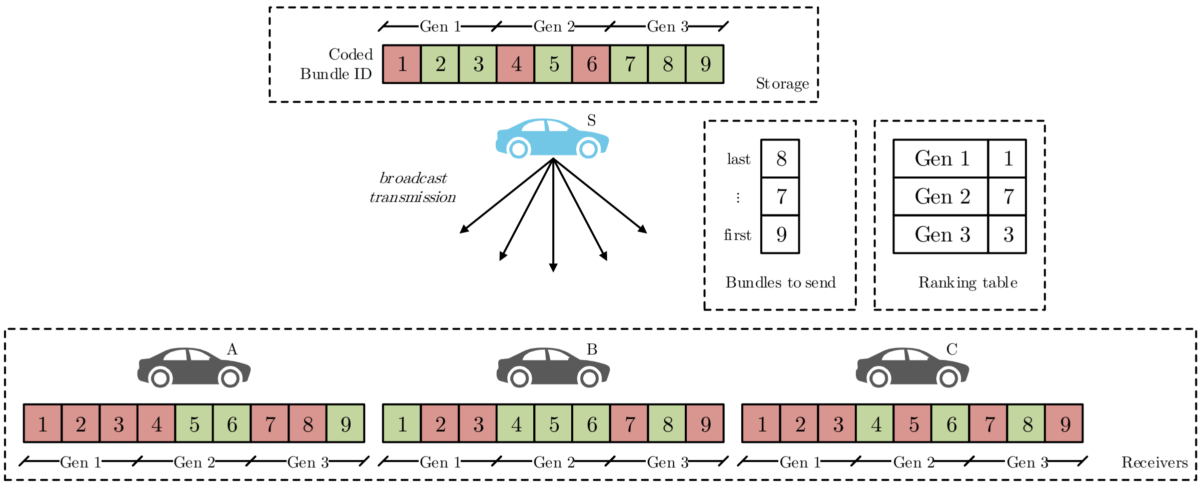


Figure 4.18: Forwarding decision of LRGF strategy - 1st situation - example

The second situation is similar to the first one and is illustrated in Figure 4.19. However, in this case the sender node can forward coded packets which were coded by another node. Thus, the sender node first sends the coded packets associated with the most lacking generation (which it already has), whereby coded packets 3 and 2 are sent first.

4.4 Strategies to Disseminate Information

The previous strategies aimed to decide which are the packets to be sent and when a node should send them based on network information collected and shared between the network

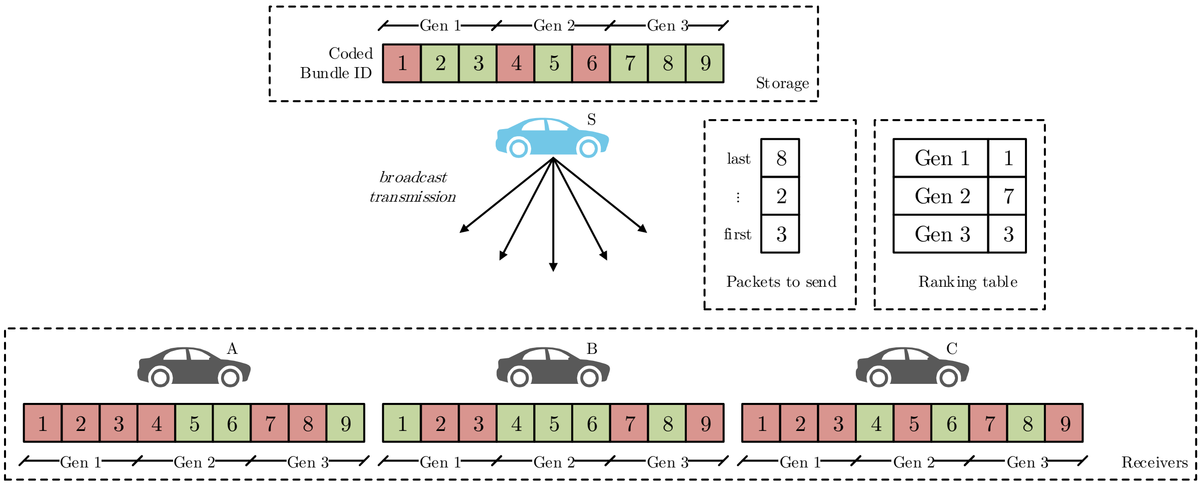


Figure 4.19: Forwarding decision of LRGF strategy - 2$^{\text{nd}}$ situation - example

nodes. Therefore, those strategies are focused on the stateless choice of information based on the current status of the network. However, several other approaches can be used to help and support those strategies in order to achieve a higher delivery rate as well as minimize the network congestion.

As an example, assuming that the medium is equally shared by K neighboring nodes, each node has access to $1/K$ of total bandwidth available to transmit data. On the other hand, the main goal of a content distribution strategy is to deliver the information as soon as possible, whereby broadcast communication is used.

Although the broadcast communication tends to increase the delivery rate, since the information is spread to a high number of nodes at the same time, it also adds new challenges related with the medium congestion. Sometimes the increase in network congestion due to the broadcast transmission produces an opposite effect leading to a decrease in the delivery rate. This last behavior tends to happen in high congestion environments where the bandwidth available is not enough for the high number of nodes trying to transmit packets in broadcast.

Due to the previous analysis, it is possible to state that it is important to establish a balance between a high delivery rate and a low medium congestion. The following subsections describe some possible techniques which are used to achieve these goals.

4.4.1 Optimize Delivery

In the strategies to stateless choose the information to send, the sending decision was based on information collected from sender nodes' neighbors (through advertisement packets) or from additional information added in the data packets. However, this decision did not consider the environment around the sender node, such as the number of neighbors within its vicinity.

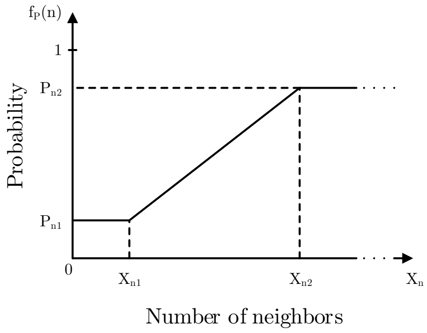
Several techniques to increase the delivery rate have been studied during this work, resulting in a set of decisions and guidelines. The first decision was to explore the broadcast nature of wireless medium by sending all data packets in broadcast. This decision aims to spread the information as fast as possible, maximizing the number of suitable receiver nodes.

However, there are other suitable approaches and techniques for delivery optimization. In this work we proposed a technique which aims to minimize the number of transmissions and maximize possible successful deliveries. The proposed technique sends information based on the sender nodes' vicinity, more precisely on the number of "valid neighbors".

Before continuing, it is important to clarify the concept of "valid neighbor". A valid neighbor is considered to be one that does not already have the content (only possible to determinate if an announcement mechanism is implemented) and has a link RSSI higher than a certain threshold (e.g. 15 dBm).

In this technique a node sends packets based on its number of valid neighbors, promoting the sending when the sender nodes have a higher number of neighbors. The main advantage associated with this approach is the fact that the amount of times that a node transmits information tends to be lower and, when it actually sends packets, these are sent to a higher number of nodes keeping the delivery rate high. On the other hand, this technique can lead to a wasting of valid contacts to send useful data to. This could be an important drawback in low density networks which typically have a low number of valid contacts per sending opportunity. Moreover, it is expected that this approach leads to high network congestion in periods with high node density (e.g. rush hours in urban areas), creating a well-known challenge called *broadcast storm.*

This approach can be deterministic or probabilistic. In the first one, a node only sends packets if it has more than X neighbors. The second one establishes an increasing probability of delivery as is shown in Figure 4.20, where a node has a specific probability of sending packets according to the number of neighbors. Although both approaches were designed and tested during the development stage, only the probabilistic one was implemented. This decision is due to the fact that the probabilistic approach tends to be less abrupt when deciding if the node should send, or not, packets in broadcast to its vicinity. Thus, using this technique a node is not forbidden to send packets when it has less then a specific number of neighbors but tends to send them less often, which achieves the main purpose of this technique without neglecting the periods with a lower number of valid neighbors.



$$f_P(n) = \begin{cases} P_{n_1} & , X_n < X_{n_1} \\ \dfrac{P_{n_2} - P_{n_1}}{X_{n_2} - X_{n_1}} X_n + \left(P_{n_1} - \dfrac{P_{n_2} - P_{n_1}}{X_{n_2} - X_{n_1}} X_{n_1} \right) & , X_{n_1} \leq X_n \leq X_{n_2} \\ P_{n_2} & , X_n > X_{n_2} \end{cases}$$

Figure 4.20: 1st probability function used in forwarding decisions

Regarding the probability function illustrated in Figure 4.20, the following considerations should be remarked:

- If the number of valid neighbor is lower than X_{n_1}, the probability of sending is defined by P_{n_1}.
- If the number of valid neighbor is larger than X_{n_2}, the probability of sending is defined by P_{n_2}.
- If the number of valid neighbors is larger or equal than X_{n_1} and lower or equal than X_{n_2}, the probability of sending is defined as an increasing function (framed between P_{n_1} and P_{n_2}), whereby the higher the number of valid neighbors, the higher the probability of sending information.

4.4.2 Minimize Congestion

As stated in the previous subsection, when the network nodes communicate through broadcast messages, a well-known challenge arrives - *broadcast storm*. This phenomenon could have a major impact on network behavior since it leads to high network congestion which has a direct impact on the content dissemination procedure. Thus, in this subsection a technique to overcome this challenge is proposed.

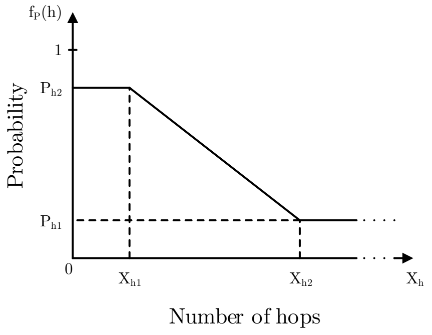
In this work, the overall concept of spreading content through a vehicular network is based on the SCF mechanism. Each time that a vehicle receives a packet belonging to a content under dissemination, it stores it, and carries this packet until it reaches another vehicle. At this moment, the node creates a copy of the packet stored and forwards it to the next vehicle, which updates the internal packet information about the number of hops where this packet already have been. Thus, this procedure is based on a multi-hop transfer of information among vehicles, whereby the information about the number of hops is directly related to the number of nodes which already have a specific packet.

Taking this into account, the number of hops can be used as a metric to estimate the overall presence of a packet (or its copies) in the network. When the number of hops is high, the probability of this packet already being stored by a considerable amount of nodes is also high. On the other hand, if the number of hops is lower, it means that this packet must be sent in order to increase its presence in the network. By enabling the sending of the most lacking packets in the network, it promotes a more selective delivery which is focused on the information that may be most lacking in the network. Besides this, it prevents that the packets travel indefinitely through the network, using bandwidth that could be used by most lacking packets.

Regarding the previous analysis and assumptions, a similar approach to the one used in subsection 4.4.1 is proposed, however, it is based on the number of hops of a specific packet instead of the number of valid neighbors. Similarly to the optimized delivery, the decision to send (or not) a packet can be deterministic or probabilistic. As shown in Figure 4.21, if a packet has 0 hops (it was not sent yet), and considering $P_{h_2} = 1$, it will be sent for sure. However, if a packet has passed through more than X_{h_1} hops, the probability of sending starts to decrease to a low value defined as the intersection of X_{h_2} and P_{h_1}, being the X_{h_1} the maximum number of hops where the probability of sending is established in P_{h_1}.

Regarding the probability function illustrated in Figure 4.21 and the proposed technique, the following considerations should be remarked:

- If the number of hops is lower than X_{h_1}, the probability of sending is defined by P_{h_2}.
- If the number of hops is larger than X_{h_2}, the probability of sending is defined by P_{h_1}.



$$f_P(h) = \begin{cases} P_{h_1} & , X_n < X_{h_1} \\ \dfrac{P_{h_2} - P_{h_1}}{X_{h_2} - X_{h_1}} X_h + \left(P_{h_1} - \dfrac{P_{h_2} - P_{h_1}}{X_{h_2} - X_{h_1}} X_{h_1} \right) & , X_{h_1} \leq X_h \leq X_{h_2} \\ P_{h_2} & , X_n > X_{h_2} \end{cases}$$

Figure 4.21: 2nd probability function used in forwarding decisions

- If the number of hops is larger or equal than X_{h_1} and lower or equal than X_{h_2}, the probability of sending is defined as a decreasing function (framed between P_{h_2} and P_{h_1}), whereby the lower the number of hops, the lower the probability of sending information.
- This approach implies the existence of an extra field in the packets header to store its number of hops.

Another technique to minimize the congestion relies on what type of nodes the sender node has in its vicinity. This technique is based on our understanding of vehicle mobility in the several scenarios in this dissertation - city scenarios - and on the kind of vehicles that comprise the network in analysis - mainly buses. As such, in an urban scenario, vehicle density tends to be fairly high in certain locations, such as main roads, city access roads, central locations, commercial areas, metro stations, train stations, among others. Besides this, the vehicles that comprise the network are mainly buses, which tend to have a well-defined route and travel through a lot of the same places, namely the more central locations, as previously listed. It is also worth noting that the buses are parked in large parking lots that can be occupied by a large number of vehicles, whereby it is in these locations that quick information dissemination occurs.

This technique aims to reduce the network congestion through the implementation of the following approach. If the sender is an OBU and it has a valid neighbor which is an RSU, it will not send the packets. This is based on the previous considerations and assumptions that claim that it is very likely that the neighbors in the vicinity of this OBU had, have or will have access to the same RSU and collect the content which is under dissemination.

An example of this is illustrated in Figure 4.22. Thus, at the beginning, vehicles A, B, and C have direct contact with an RSU which always has the content under dissemination available. So, all the three vehicles have access to the same RSU which is broadcasting a certain content, whereby all the vehicles are listening the content from the fixed infrastructure. Although vehicles A, B, and C are in the range of each other, due to the application of this technique there is no communication among them, which, as previously clarified, is not necessary. Later, when vehicle C reaches other vehicles (D and E), it will broadcast the content previously collected from the RSU.

It is important to announce that this technique is implemented in all of the experiments and evaluation procedures presented in Chapter 6.

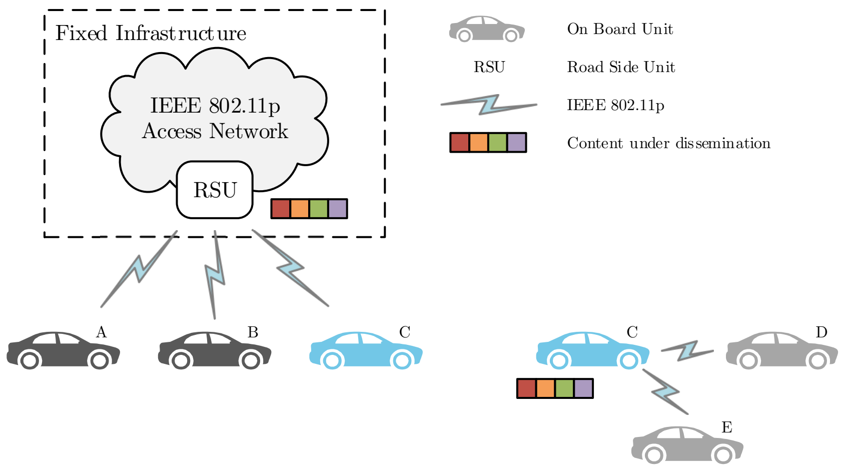


Figure 4.22: Technique to minimize congestion based on type of node

4.4.3 Hybrid Technique

In the previous subsections two kinds of techniques were suggested to optimize the delivery and to minimize the network congestion. However, in a content dissemination service both of these enhancements are desirable, whereby in this work a hybrid technique is suggested using these two already proposed approaches.

In both previous techniques a probabilistic approach is proposed in order to minimize the undesirable effects brought by network congestion and enhance a higher delivery rate. Thus, this hybrid technique aims to combine these two techniques by multiplying their characteristic probability functions in order to produce a new one that reflects the main goals for which they were previously designed. Assuming that both probability functions are independent and uniformly distributed, a new probability of sending is evaluated by multiplying the previous two: $f_p(n_{neighs}, n_{hops}) = f_p(n_{neighs}) \times f_p(n_{hops})$. Figure 4.23 illustrates a possible output of this function and is the multiplication result of functions shown in Figure 4.20 and Figure 4.21.

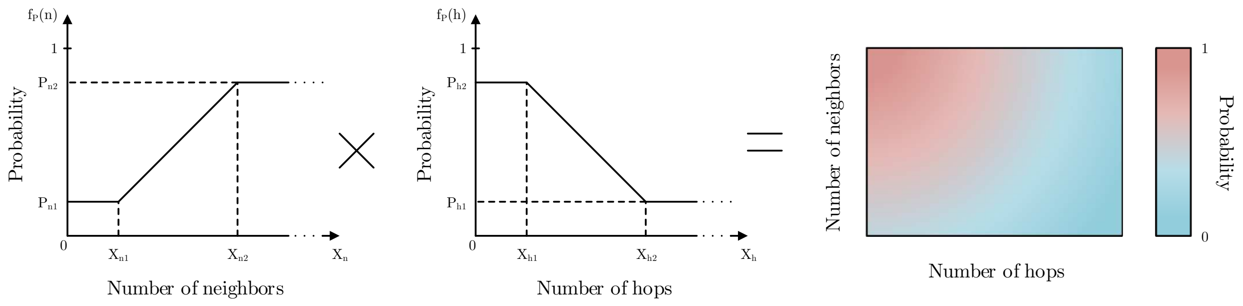


Figure 4.23: 3rd probability function used in forwarding decisions

With the introduction of this new technique, it is expected that the advantages associated with both aforementioned techniques are obtained. As such, an improvement in delivery rates due to choosing the best scenarios for sending packets (when there are more neighbors) is expected, as well as the minimization of network congestion since there will not be any packets travelling for an unnecessary or excessive period of time.

4.5 Chapter Considerations

As identified in Chapter 1, the main goal of this work is the study, design and development of content distribution schemes for non-urgent content dissemination. Thus, this Chapter was focused on the description of the proposed schemes. Two different sets of strategies were described: (i) Strategies to Stateless Choose Information and (ii) Strategies to Disseminate Information. The implementation of these strategies is performed in Chapter 5 and their performance is evaluated in Chapter 6.

Strategies to Stateless Choose Information

The first set of strategies is mainly focused on the selection of which packet should be sent. Four strategies was proposed: (i) Random, (ii) LNHF, (iii) LRBF, and (iv) LRGF.

As the name implies, in the Random strategy, the packets to send are randomly selected from the node's storage. This strategy aims to perform a forwarding decision without adding any additional control data in order to keep the network overhead as low as possible.

The next proposed strategy was the LNHF. This strategy performs the forwarding decision using one of two parameters associated with each packet: number of hops and number of transmissions. The number of hops represents the number of nodes which has stored this packet (or a copy of it), whereby it can be used as a metric to understand how traveled is the packet. The higher the number of transmissions, the higher is the probability of this particular packet to be already stored by a higher number of nodes. Moreover, this decision depends on the sender node type. In an RSU the number of hops can not be used as a parameter to decide which packet should be sent, since all packets have the same number of hops (assuming that RSU is directly connected to the content server). Thus, in the RSUs the used criteria is the number of transmissions. On the other hand, the OBUs use the number of hops as main criteria and the number of transmissions as a tiebreaker criteria. This decision is due to the fact that OBUs have access to packets with different number of hops and this criteria is a better indicator of the packet's mobility.

The LRBF strategy is the first proposed strategy which aims to perform a forwarding decision based on the sender node's vicinity storage content. Thus, the sender node has to identify which is the most lacking packet within its vicinity. To perform this evaluation, the addition of advertisement packets is proposed. These packets periodically advertise the node's storage content, indicating which packets and contents they have stored. Once collected the information about all the neighbors, the sender node can evaluate which are the most lacking packets. Then, if the sender node has this packet stored, it can broadcast it to its vicinity. Several considerations must be taken into account in the implementation of this strategy. It is very important to establish a compromise between the periodicity of these advertisement packets and the network overhead. The frequency of spreading advertisement packets introduces more accuracy in the forwarding decision; however, due to the foreseeable size of the advertisement packet, the network overhead can be considerably high.

The last strategy is proposed to be used in a network which has applied the network coding concept. This strategy is focused on the selection of which coded packet should be forward. In order to perform this decision, it is also proposed the use of advertisement packets which should broadcast a parameter called rank. This parameter aims to identify which are the most lacking block/generation in the sender node vicinity, and it is evaluated based on the number of coded packets stored associated with a specific block/generation that a node has. Although it uses advertisement packets, it is expected a lower network overhead when this strategy is used compared to the LRBF approach. However, the advertisement packet size is closely related with the *blocksize* parameter, which indicates the number of frames of a block (the content is divided in blocks). On the other hand, it is also expected a slower progress in the content distribution procedure (when compared to the LRBF strategy), since a node needs to receive a minimum number of packets (*blocksize*) to decode a complete block.

Strategies to Disseminate Information

The previous strategies are focused on the stateless choice of information based on the current status of the network. On the other hand, other strategies can be proposed to support those strategies in order to minimize the network congestion and overhead along with optimize the delivery rate. To enhance a fast content dissemination, all communications are performed in a broadcast manner; however, this kind of approach leads to a high network congestion which sometimes produces an opposite effect leading to a decrease in the delivery rate. Thus, it is important to establish a balance between a high delivery rate and lower medium congestion, whereby a set of techniques are proposed in order to achieve these goals.

The technique proposed for delivery optimization aims to minimize the number of transmissions and maximize possible successful deliveries. The proposed technique sends information based on the sender nodes' vicinity, more precisely on the number of "valid neighbors" (connection's link has a specific minimum power). When this technique is applied, a node sends packets based on its number of valid neighbors, promoting the sending when the sender nodes have a higher number of neighbors. The main advantage of this strategy is the lower amount of times that a node transmits information, and when sent, they tend to reach a higher number of nodes keeping the delivery rate high. However, this approach can lead to a wasting of valid contacts and can increase the network congestion due to the high number of vehicles involved in the transmission instant. The last issue can create a well-known challenge called *broadcast storm*.

In order to mitigate the last identified challenge, a technique to overcome this challenge was proposed. The content dissemination procedure is based on a multi-hop transfer of information among vehicles, whereby the information about the number of hops is directly related to the number of nodes which already have a specific packet. Thus, the number of hops is used as a metric to estimate the overall presence of a packet (or its copies) in the network. This technique promotes the sending of packets with a lower number of hops in order to optimize the delivery of the most potential lacking packets. Moreover, it prevents that the packets travel indefinitely through the network, using bandwidth that could be used by most lacking packets. Another technique is proposed to minimize the network congestion, and is focused on the neighbor type. If the sender node has a neighbor which is an RSU, it is not able to forward packets because probably the majority of its neighbors had, have, or will have access to the same RSU.

At the end, a hybrid technique is presented with the main goal of achieving the last to premises: optimize delivery and minimize the network congestion. This technique is implemented as a function of the two previously proposed techniques.

Chapter 5

Integration and Development

5.1 Chapter Description

Once the content distribution strategies are identified and designed (high-level), they need to be developed and integrated in a set of platforms. Three platforms are used during this Dissertation: a Matlab-based emulator, an emulator specifically designed to develop and perform scalability tests of the DTN mechanisms, and finally the vehicular OBUs are used in a laboratory environment. Thus, the proposed strategies are developed and implemented in order to be evaluated on the previous platforms.

This chapter is organized as follows:

- *section 5.2 - MatlabEmulator Development and Integration*: describes the implementation of a Matlab-based emulator used to easily evaluate several strategies for content distribution.
- *section 5.3 - HelixEmulator*: describes an emulator especially designed to perform scalability tests of the DTN mechanisms along with a set of improvements and modifications performed under the scope of this work.
- *section 5.4 - Helix Integration*: presents the main modifications performed in the DTN HelixEmulator in order to implement the proposed content distribution strategies along with the major challenges overcame in the process.
- *section 5.5 - Vehicular OBUs Integration*: describes the modifications and considerations taken into account regarding a smooth transition from an emulation environment of the HelixEmulator to the real network boards.
- *section 5.6 - Chapter Considerations*: depicts the conclusions and the summary of the full chapter.

5.2 MatlabEmulator Development and Integration

5.2.1 Introduction

As concluded before (see section 3.4), several mechanisms can be used to emulate a network and test a variety of protocols and strategies. However, none of them are ideal since each has its singularity, being more or less complex and applicable to the proposed study. Compared to the trivial or widespread solutions previously listed (see section 3.4), in order to perform this study no mobility model is needed since a data log set collected from a real network is available. Thus, the decision was to build a network emulator from the start in order to emulate the content dissemination process using the collected information of the mobility of vehicles.

In a previous work developed by Ricardo Matos from Veniam$^{©}$, an emulator was developed and implemented in MATrix LABoratory (MATLAB)$^{®}$ [155]. However, this emulator only allows the dissemination of information in an upstream way, and so it was not able to run and test a content dissemination process started by a server located in the core network to the vehicles (OBUs) (downstream).

Thus, the decision was to implement a new emulator based on the previous one but which allows multi-hop traffic in a downstream way and that implement the proposed dissemination strategies. Within this document, it is called MatlabEmulator. Along this section, its architecture is explained and all of its characteristics are discussed.

5.2.2 Architecture and Operation Overview

As illustrated in Figure 5.1, the MatlabEmulator architecture can be divided mainly in four stages/phases named from 0 to 3. They are described below.

In the first stage the configuration of the emulator and experiment is selected. The content distribution scheme is defined here, as well as the log information source. Furthermore, the experiment is also configured, parameters such as size file and scheduling of its dissemination are also defined here.

The second phase is responsible for uploading the log information previously collected and stored in log files into context variables used to perform routing decisions during the emulation process.

The third phase is the main stage of the emulation process. Within this stage the multi-hop routing decisions (transmission process) for dissemination, defined as strategies to stateless choose information in Chapter 3.3, are made. After that, the reception process (particularly the bandwidth control) is emulated. The strategies to disseminate information, described in Chapter 3.3, are also implemented during this stage. Moreover, output variables used for statistical analysis in the last phase are updated.

Finally, in the last stage, the statistical analysis of the emulation process using the output variables resulting from the third stage is done. A more refined explanation of these stages is done is the next sections.

Although the emulator operates in a sequential way, and the phases 1 to 3 are dependent on each other and are all related to the configuration (phase 0), they can be executed separately. At the end of stages 1 and 2, the data is stored into MATLAB files and can be updated by the following stages at their beginning. All these options are configured at phase 0. This feature was introduced because most of the times the context variables are common among multiple experiments, since they are only related with the dataset of log information used, not varying with the content distribution scheme selected. Moreover, the first stage can be time consuming, whereby the possibility of running this stage separately improves time efficiency.

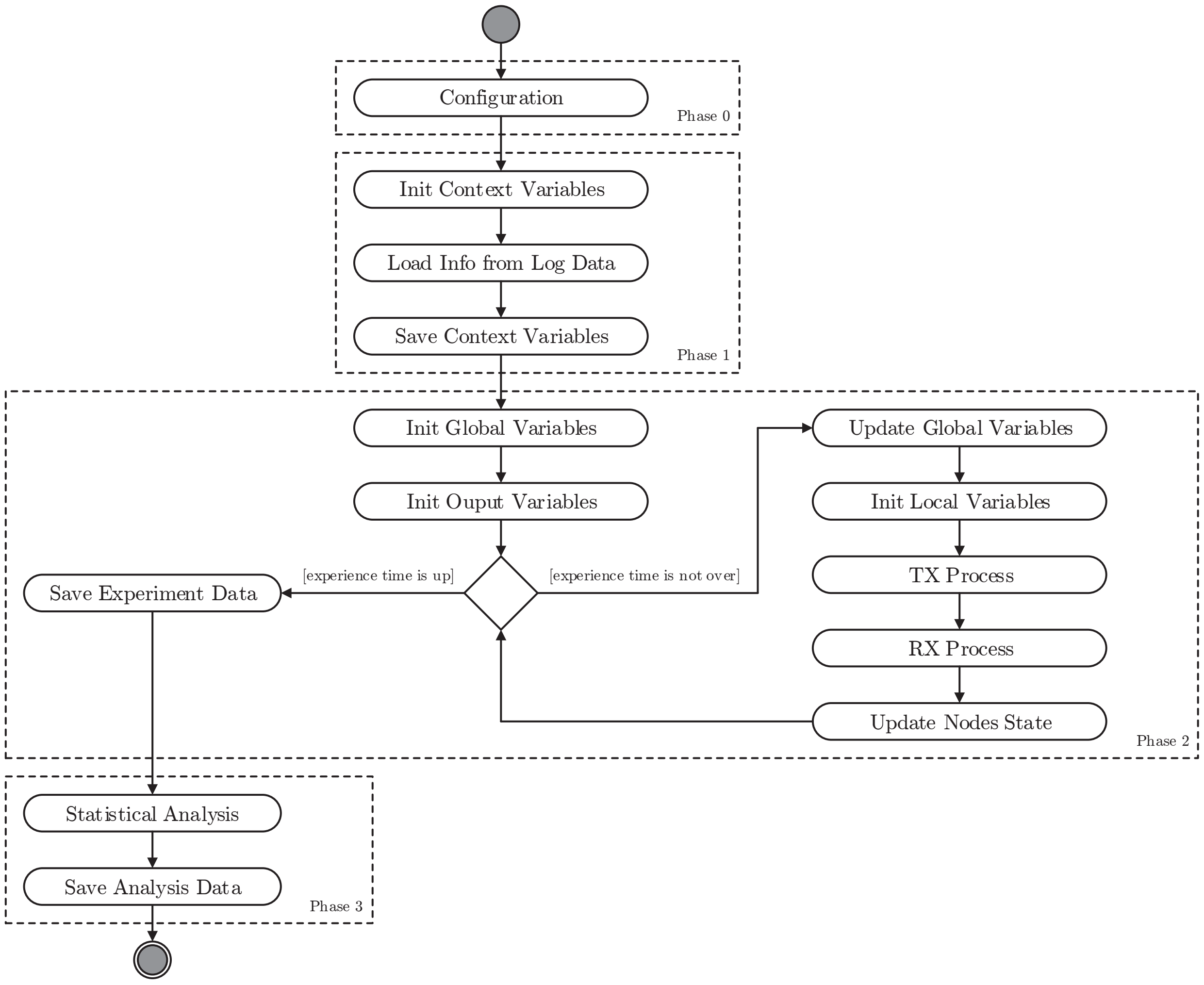


Figure 5.1: MatlabEmulator architecture and operation flowchart

5.2.3 Collected Log Data

Before describing the collection of log data procedure and the MatlabEmulator itself, it is important to understand the log files structure. During these past months, two sets of log data were collected during a 24-hour period in the vehicular testbed under the scope of this work [8]. Both structures are roughly equal and only differ on a few extra fields added in the last dataset of log files but with no influence in the resulting context variables. So, the structure of the most recent set of log files is described in the following sections.

A log information is stored as a Comma-Separated Values (CSV) file, and its structure is described in Figure 5.2. The main fields used in the evaluation of context variables are described as follows. The additional information (e.g. cell and cell RSSI) can be used to create and evaluate new context variables in the future.

- *Node ID*: this field allows the identification of the node where the log information was generated.

- *UNIX Timestamp*: identifies the timestamp of when the log data was collected.
- *GPS coordinates*: indicates the geographical position of the node at this timestamp.
- *Number of Neighbors*: number of neighboring nodes at the indicated timestamp.
- *Link info*: this pair of values (ID,RSSI) specifies the link properties for each neighboring node.

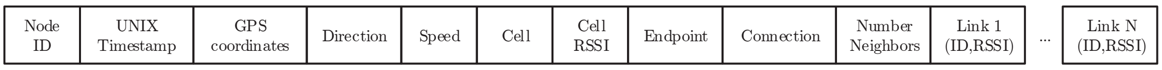


Figure 5.2: MatlabEmulator log file structure

5.2.4 Working Variables

MATLAB is based on matrix manipulation, whereby all variables are represented by a matrix of one or more dimensions. As mentioned before, the used variables can be classified in four groups/categories: *context*, *global*, *local*, and *output*. They are described as follows.

The **Context** variables are evaluated from the collection log data procedure, and provide a set of parameters which allow the evaluation of the identification of nodes, their geographical position at a certain time instant, their list of neighbors, among other characteristics. The following list represents the resulting variables.

- `nodes`: vector containing a list of OBUs present in the log dataset.
- `allnodes`: vector containing a list of OBUs and RSUs present in the log dataset.
- `timestamps`: vector containing all the UNIX timestamps present in the log dataset.
- `timesvalid`: binary matrix which indicates if any log data was collected about a specific node at a certain timestamp.
- `rssineighbors`: three dimensional vector which specifies, for a specific timestamp, the RSSI link value of a connection between an OBU (or RSU) and other OBU.
- `distneighbors`: three dimensional vector which specifies, for a specific timestamp, the distance (in meters) between an OBU (or RSU) and another OBU.
- `gpscoordinates`: matrix containing the GPS coordinates (latitude and longitude) of a certain node (OBU or RSU) at a specific timestamp.
- `RSSI_MIN`: minimum link connection RSSI to ensure a stable and reliable communication between two nodes. If an RSSI is greater than `RSSI_MIN`, a null packet loss for that connection is assumed.
- `Ta`: sampling time of log data, it can assume one of the two values, 5 or 2 seconds, depending on the `DATASET` selected.
- `BUNDLE_SIZE`: bundle size in bytes (default value of 32 KB).

- MAX_BUNDLE_SECOND: maximum number of bundles that a node can receive (or transmit) during a sampling time according to a default bandwidth value of 1 Mbps and a BUNDLE_SIZE of 32 KB.

As a remark, the BUNDLE_SIZE is defined based on the Helix data packet's maximum size (which is 32 KB) and the default bandwidth is defined to be a small percentage of the maximum available bandwidth in the NetRider boards (which is 27 Mbps), since the proposed content distribution service is not a high-priority service.

Global variables emulate the storage module of a node (OBU or RSU) and retain the state of a bundle (or coded bundle). They are critical to perform forwarding procedures since they establish if a bundle is available (or not) to be disseminated. Moreover, they retain other characteristics of each bundle such as number of hops, number of transmissions, or, in case of a coded bundle, the generation to which they belong.

- bundles_stored: three dimensional vector which stores the state of a bundle in a specific node (OBU or RSU) at a specific timestamp. The node's state is composed by three components, (i) availability (never was available, is available, or it is no longer available), (ii) number of hops, and (iii) number of transmissions.

- codbundles_stored: is similar to the previous one, but it represents a coded bundle stored by a node, instead of a bundle. Furthermore, the second dimension is no longer the number of hops, but the generation to which the coded bundles belong.

The **Local** set of variables are reseted at each iteration of the emulation procedure, and they store instant information about the bundles (or coded bundles) to be forwarded, neighbors of each node, and what is the file which is being spread.

- bundles_to_send: this matrix stores the number of bundles (or coded bundles) that each node (OBU or RSU) will try to send to other nodes (OBU).

- bundles_already_sent: local matrix, inside the transmission process, that counts the number of bundles (or coded bundles) already sent by a node (OBU or RSU).

- nodes_pernode: this matrix stores the RSSI link of each intended connection between two nodes at this timestamp.

- indexs_bundles: is a two dimensional vector which stores the indexes of bundles selected by a node to send (ordered according to the content distribution scheme) at this timestamp.

- file_idx: this matrix identifies the file which is being disseminated at this timestamp.

Output variables are used in stage 3, when the statistical analysis is performed. The resulting matrices are described below.

- broadcast_opportunities_total: number of times that a node (OBU or RSU) wants to contact its neighbors (OBUs) per timestamp.

- broadcast_opportunities_used: number of times that a node (OBU or RSU) effectively contacts its neighbors (OBUs) per timestamp.

- `bundles_pernode`: number of bundles (or coded bundles) stored per node (OBU or RSU) per timestamp.
- `files_completed`: stores the timestamp when the node (OBU) has received a completed file, or zero if it has not.
- `bundles_received_bad`: number of bundles (or coded bundles) that a node (OBU) received and already had per timestamp per node.
- `bundles_received_good`: number of bundles (or coded bundles) that a node (OBU) received and did not already have per timestamp per node.
- `bundles_rejected_bad`: number of bundles (or coded bundles) that were rejected (due to bandwidth limitation) by the destination node (OBU) and were missing per timestamp per node
- `bundles_rejected_good`: number of bundles (or coded bundles) that were rejected (due to bandwidth limitation) by the destination node (OBU) and were not missing in destination node per timestamp per node.
- `bundles_listened`: number of bundles (or coded bundles) heard in the medium per timestamp per node (OBUs).
- `bundles_transmitted`: number of bundles (or coded bundles) transmitted (in broadcast) per timestamp per node (OBUs and RSUs).

5.2.5 Phase 0 - Configuration

The process of configuration starts with the definition of macro variables. They are responsible for setting the main characteristics of the data collection uploading, simulation, and statistical analysis procedures. The description is done below.

- `DATASET`: sets the collection of log data to be used by the emulator. The following values can be defined:

  `2` - Dataset of October 2014 (see section 6.4).

  `3` - Dataset of February 2015 (see section 6.4).
- `FILTER_NODES`: defines if all nodes will be emulated or only a few based on smaller lists which only contain a specific set of nodes (for example based on their geographical location along the experiment).
- `FILTER_NODES_TI` and `FILTER_NODES_TF`: when `FILTER_NODES` is enable these macros define the start and the end of emulation, and also what is the sub-dataset (previously evaluated based on a specific criteria) that will be used..
- `RUN_PHASE_X`: it is divided into three variables (phases I, II, and III), and allows the separate execution of different stages (as mentioned before, see section 5.2.2).
- `LIMITED_LIFETIME`: defines the lifetime of bundles, particularly it defines if a bundle (or a set of bundles) will continue to be forwarded by OBUs after the end of availability of the file, to which they belong, in RSUs.

- `LIMITED_RX_FILTER`: restrain the reception bandwidth limitation only to a specific period of time, which can be defined in the emulator (default is between 6am and 12pm).
- `LIMITED_REPLICAS`: restrain the forwarding of bundles by the OBUs (replicas) only to a specific period of time, which can be defined in the emulator (default is between 6am and 12pm).
- `RSU_STATIONs`: allows (or not) the introduction of two specific RSUs, which are located in the parking lot of the transportation company, in the emulation process.
- `REPLICAS`: allows (or not) the forwarding of bundles by the OBUs (replicas), implementing a multi-hop transmission.
- `RX_FILTER`: allows (or not) the reception bandwidth limitation to a specific value, which can be defined in the emulator (default is 1 Mbps).
- `TX_VERSION`: defines the transmission/forwarding process characteristics, mostly the limitation related with the optimized delivery and minimized congestion. The following values can be defined:

  `1` - Deterministic approach based on a neighboring threshold (send if number of neighbors $\leq$ `NEIGH_THRESH`).

  `2` - Deterministic approach based on a neighboring threshold (send if number of neighbors $\geq$ `NEIGH_THRESH`).

  `3` - Deterministic approach based on a hops threshold (send if number of hops l `MAX_HOP`).

  `11` - Probabilistic approach based on a decreasing probability function of number of hops as illustrated in Figure 4.21.

  `12` - Probabilistic approach based on an increasing probability function of number of neighbors as illustrated in Figure 4.20.

  `13` - Probabilistic approach based on an hybrid solution described in Figure 4.23.
- `ORDERING_VERSION`: defines the strategies to stateless choose information (random, LNHF, LRBF, and LRGF content distribution schemes), and some variants of each. The following values can be defined:

  `0` - Enables Random strategy.

  `2` - Enables LNHF strategy.

  `1` - Enables LRBF strategy.

  `11` - Enables LRGF strategy - 1st situation (see subsection 4.3.4).

  `11` - Enables LRGF strategy - 2nd situation (see subsection 4.3.4).
- `CODING`: if the coding approach (LRGF content distribution scheme) is active, this macro is enabled.

After the "main" macros, the "auxiliary" ones are defined. The first set is related to the chosen `TX_VERSION`. If the transmission version is a deterministic one, the following auxiliary macros will be defined:

- `NEIGH_TRESH`: defines a threshold related with the number of neighbors of a transmitter node. If the instant number of neighbors is lower (or greater) than this value, the node will forward bundles to its neighbors.
- `MAX_HOP`: defines a maximum number of hops that a bundle can "travel" through the network. The dissemination of a certain bundle will stop when it reaches this value.

On the other hand, if it is a probabilistic approach, the following auxiliary macros will be defined:

- `X1`, `X2`, `P1`, and `P2`: these macros are related with the definition of the probability functions illustrated in Figures 4.20, 4.21, and 4.23.

If a probabilistic approach which involves limitation based on the number of neighbors and number of hops is defined, two new pairs of (`X1`,`P1`) and (`X2`,`P2`) are defined, to establish the probabilistic functions related to both types of limitations.

Next, the auxiliary macros related to the file to be spread are defined. The three main settings related to it are the number of files to be spread and their size, and the dissemination periods of each one. So, the following auxiliary macros have been defined:

- `F_SIZE`: defines the size of the files (in number of bundles).
- `i_F_BEGIN` and `i_F_END`: define the beginning and ending of each file spreading.

Finally, if the coding (LRGF content distribution scheme) is active, the two main parameters are the following:

- `BLOCKSIZE`: defines the number of bundles within a block (it has to be a multiple of `F_SIZE`).
- `GENSIZE`: defines the maximum number of coded bundles that can be generated from a certain block.

5.2.6 Phase 1 - Collection of Log Data

Figure 5.3 illustrates the data collection process. The origin of the log information is OBUs, although there is a need to create information about the RSUs neighbor list and their links, whereby a backtracking approach is used to generate such log files. As an example, if OBU A has as neighbor RSU B with a link's RSSI of X dBm, thus RSU B also has a neighbor A with a link's RSSI of X dBm. According to the `RSU_STATIONs` macro, which establishes the use (or not) of the RSUs of the collection stations in the emulation process, a list of RSUs to be used is generated. Since the two datasets of log information collected and used by the emulator presented different sampling time collection of 5 and 2 seconds, according to the `DATASET` macro, the sampling time `Ta` (or emulation time step) is defined. After that, the `BUNDLE_SIZE` (in bytes) is defined and the default value is 32 KB. Based on the previously define value, the bandwidth limitation (in number of bundles per second), `MAX_BUNDLE_SECOND`, is evaluated. One important parameter is the minimum RSSI to have a stable connection which guarantees the delivery of information between nodes without any packet loss. The default value of this variable is established as 15 dBm.

If there is any file in the log folder, the *Node ID* and *UNIX Timestamps* are collected while there are still logs to read. Moreover, the `nodes`, `allnodes` and `timestamps` variables are updated. Furthermore, if FILTER_NODES is enabled, the nodes and RSUs vectors are updated with a specific predefined set of nodes. After that, the context variables are initiated, and all the log files are read to update them, particularly the timesvalid, rssineighbors, and gpscoordinates vectors.

Since the data collection only has information about OBUs, the RSUs statical geographical location is introduced manually. Using this information and the previously collected OBUs location, the distance between nodes is evaluated and stored in the distneighbors variable.

A set of neighboring statistical analysis is performed at this stage, information about how many neighbors, what type of nodes they are, if they are valid neighbors (link RSSI greater than RSSI_MIN), among others, is stored in several variables. This information is useful to evaluate global metrics associated with an overview study of the network behavior that can be used to define the transmissions parameters.

Finally, the evaluated context variables are saved to be used in the following stages.

5.2.7 Phase 2 - Emulation

This is the main stage of the emulator since it performs the emulation process itself. At the beginning, the global and output variables are initiated. After this initial operation, a sequential process starts which extends until the experience time is up (see Figure 5.1). This last iterative process is described in detail in the following sections.

5.2.7.1 Update and Init Variables

The global variables are mutable during the emulation process since the files have a specific dissemination time interval. So, the bundle_stored (or codbundles_stored) needs to be uploaded during the process. When the file under dissemination changes, these variables update the bundle state in storage into *not available* (if it was available in the past), whereby the forwarding of such bundles is no longer possible.

Moreover, if the LIMITED_RX_FILTER or LIMITED_REPLICAS macros are active, the RX_-FILTER and REPLICAS macros, respectively, need to be updated.

5.2.7.2 Transmission Process

The transmission process flowchart is presented in Figure 5.4.

As the emulator was developed to emulate the content dissemination of a file, the traffic flow is downstream (from local servers to endpoints). Thus, since only two types of nodes are considered (OBUs and RSUs), the file is considered to be instantly available in the RSUs and remained stored during the whole dissemination process, and there are only two possible link flows: (i) from RSUs to OBUs, and (ii) from OBUs to other OBUs. Once the transmission of a bundle can start in both OBUs and RSUs, the iterative transmission process runs across the allnodes vector. While there are still nodes to emulate, the transmission process is emulated.

If the timestamp is valid (log data was collected about this node in this timestamp), the neighbor list is derived. This list consists in a set of pairs (neighbor ID, link RSSI) and is evaluated based on the context variable rssineighbors. Since the destination node is always an OBU, the RSUs are excluded from the neighbors list. Furthermore, only valid neighbors (link's RSSI greater than RSSI_MIN) are presented within this list. Finally, if a node has a valid RSU in its vicinity, the neighbors list is empty since there is a high probability of node neighbors also having contact with this same RSU. This last modification has the goal

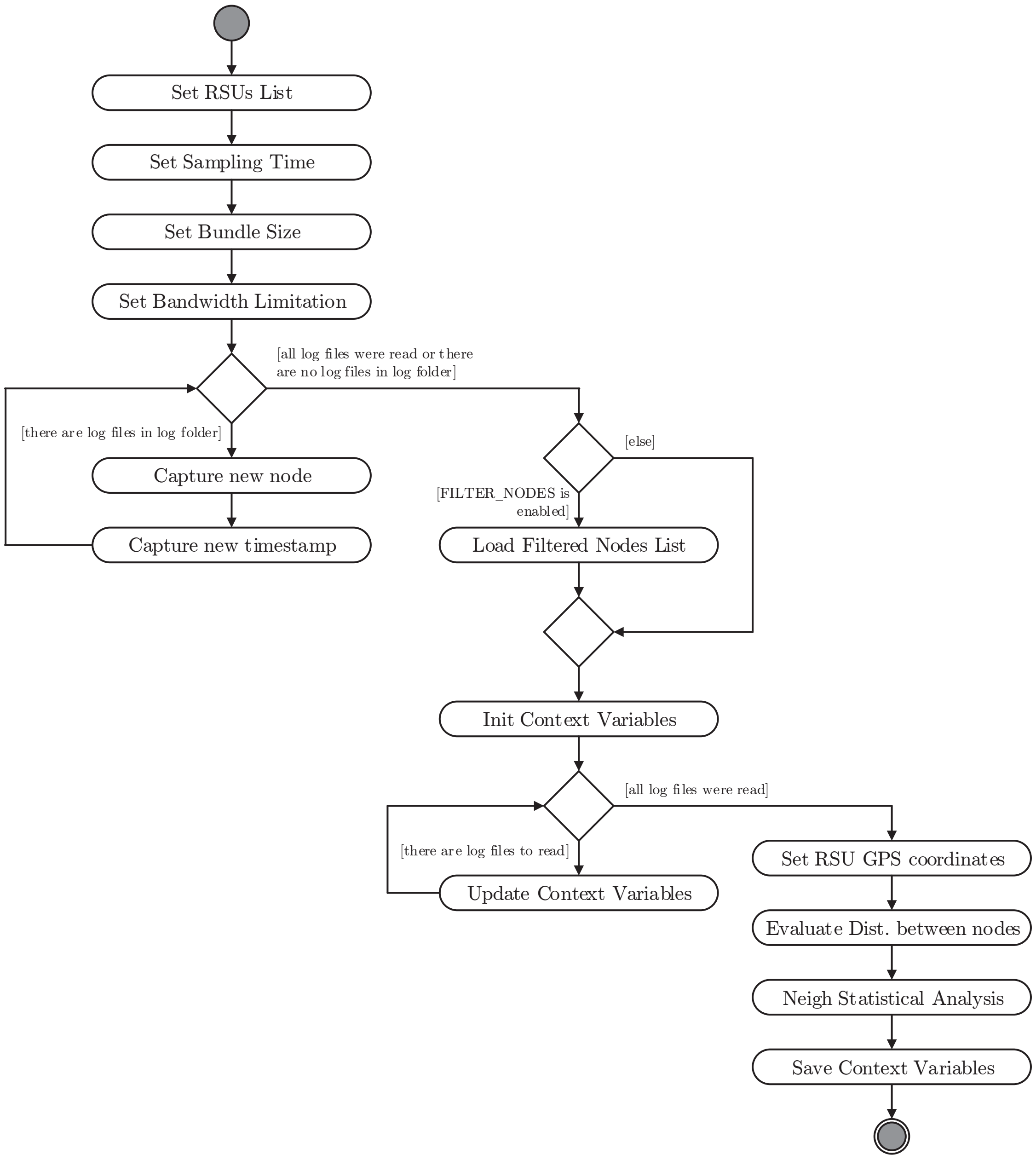

Set RSUs List
Set Sampling Time
Set Bundle Size
Set Bandwidth Limitation
[all log files were read or there are no log files in log folder]
[there are log files in log folder]
Capture new node
Capture new timestamp
[else]
[FILTER_NODES is enabled]
Load Filtered Nodes List
Init Context Variables
[all log files were read]
[there are log files to read]
Update Context Variables
Set RSU GPS coordinates
Evaluate Dist. between nodes
Neigh Statistical Analysis
Save Context Variables

Figure 5.3: MatlabEmulator data collection (phase 1) process flowchart

of optimizing the delivery (since the RSU already has all of the file), and minimizing the congestion (since this node will not forward any bundle within this time interval).

According to the previously defined `TX_VERSION`, some restrictions can be applied. The neighbors list is empty (which invalidates the continuation of the transmission process) in the following conditions related with the number of neighbors:

- *Deterministic approach*
  - `TX_VERSION` has the value `1` (or `2`) set, which implies a limitation based on the

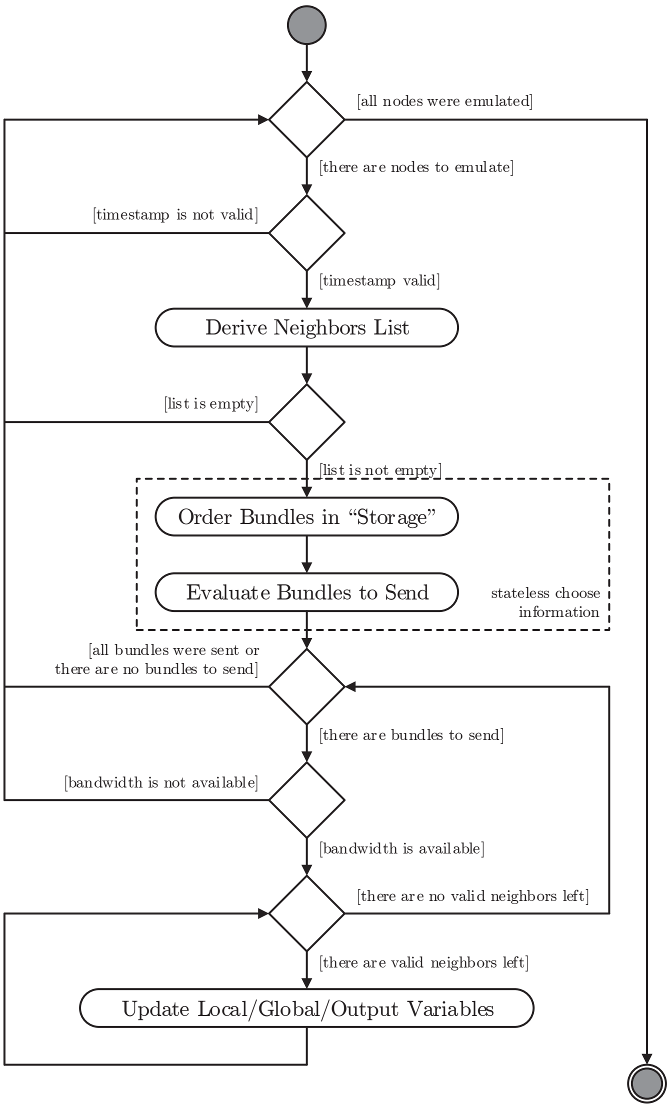

[all nodes were emulated]
[there are nodes to emulate]
[timestamp is not valid]
[timestamp valid]
Derive Neighbors List
[list is empty]
[list is not empty]
Order Bundles in "Storage"
Evaluate Bundles to Send
stateless choose information
[all bundles were sent or there are no bundles to send]
[there are bundles to send]
[bandwidth is not available]
[bandwidth is available]
[there are no valid neighbors left]
[there are valid neighbors left]
Update Local/Global/Output Variables

Figure 5.4: MatlabEmulator transmission process flowchart

number of neighbors, and the number of neighbors is larger (or lower) than the threshold defined by `NEIGH_TRESH`

- *Probabilistic approach*
  - `TX_VERSION` has the value `12` set, which implies a limitation based on the number of neighbors, and the probability of sending (evaluated based on `X1`, `X2`, `P1`, and `P2` macros) is lower than a uniformly generated random number between 0 and 1. A more detailed description of this probability function is in section 4.4.

Once the neighbor list is derived, and if the list is not empty, the emulator proceeds to the "storage" ordering. The content distribution scheme is defined by setting the `TX_VERSION` and `ORDERING_VERSION` macros. The latter is responsible for establishing the strategy to stateless choose information (bundles) from the storage. How that choice is made is described below.

The first approach (`ORDERING_VERSION=0`) is the Random selection of bundles within the stored and available bundles (identified with `1` in `bundles_stored` matrix). Thus, the storage is organized based on the resulting randomly ordered vector of bundles.

If LNHF is the selected content distribution scheme (`ORDERING_VERSION`=2), the bundles are ordered in a different way depending on the type of sender node. If it is an RSU, the storage is ordered according to the number of transmissions: the bundle with a lower number of transmissions is the first to be disseminated to other vehicles. On the other hand, if the sender node is an OBU, the first bundle in storage is the one with a lower number of hops. The number of transmissions is obtained from the `bundles_stored` matrix, which has the number of times that a bundle has been broadcasted stored in one of its dimensions.

The LRBF strategy is selected setting the `ORDERING_VERSION` macro to `1`. In this approach, the bundles lacking on the node's valid neighbors are evaluated using `bundles_-stored` vector. Once the bundles lacking are evaluated, the bundles stored by the sender node are ordered according to the most common bundles lacking in its neighbors, being the first bundle in storage the most lacking bundle in the node's valid neighbors that the sender node has.

Finally, if the selected content distribution scheme is the LRGF, three different approaches are implemented. Nevertheless the implementation of the announcement of neighbors ranking is similar among them. This procedure is illustrated in Figure 5.5. First, if there are valid neighbors with content (coded bundles) in their storage, the ranking is evaluated. The ranking is the number of coded bundles within their storage, truncated to a maximum value of `BLOCKSIZE`. The ranking is evaluated for all valid neighbors and all generations of the file. This process results in two vectors containing the accumulated ranking for each generation and the number of times that information about a certain generation was received.

After collecting the rankings, and if any was collected, the storage is ordered according to the selected version of LRGF content distribution scheme. All versions order the storage according to the most common generations lacking, however, there are small differences among them. If the version `11` was selected (1st situation in subsection 4.3.4), only coded bundles associated with the already decoded blocks (at least `BLOCKSIZE` coded bundles were received) are mapped in storage and available to be forwarded. On the other hand, if the version `13` is selected (2nd situation in subsection 4.3.4), coded bundles belonging to non-decoded blocks can also be mapped in storage and, thereafter be forwarded by the sender node. In the two previous versions, the storage is mapped in groups of `BLOCKSIZE` coded bundles. In other words, the first `BLOCKSIZE` coded bundles in the storage belong to the most lacking generation, the second set of `BLOCKSIZE` coded bundles belong to the second most lacking generation, and so on.

All the previous ordering strategies aim to put at the head of storage the most suitable set of bundles (or coded bundles) according to the selected content dissemination scheme. Thus, the evaluation of bundles (or coded bundles) to send is simply done by scrolling the map of storage iteratively. However, some transmission versions (indicated by `TX_VERSION` macro) impose a filtering of the pre-selected bundles to send based on the number of hops of each bundle. Thus, as described in section 4.4, there are two filtering scenarios: limitation by (i) maximum number of hops using a deterministic approach imposed by `MAX_HOP` macro, or (ii) using a probabilistic approach imposed by a probability function illustrated in Figure 4.21.

Once the bundles (or coded bundles) to send are evaluated, an iterative process which is responsible for updating local, global and output variables begins. This procedure emulates the intention of sending a bundle by the sender node, in broadcast, to all its neighbors. A

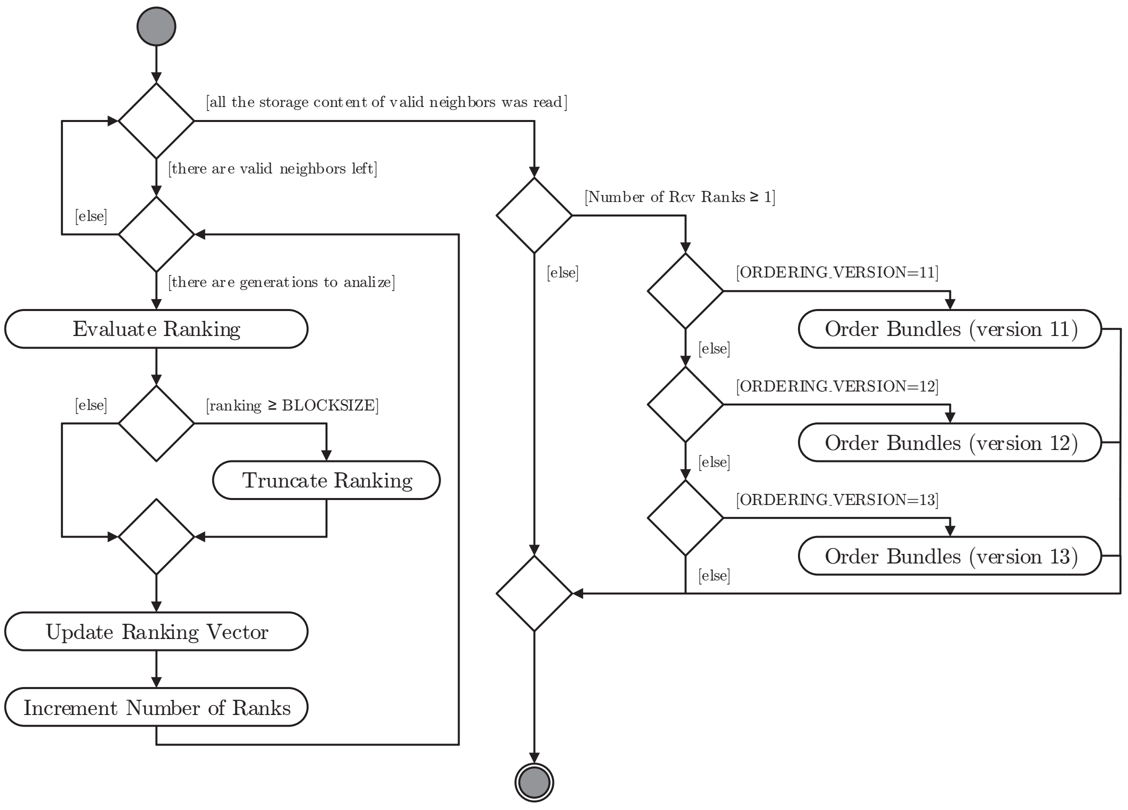

[all the storage content of valid neighbors was read]
[there are valid neighbors left]
[else]
[there are generations to analize]
Evaluate Ranking
[else]
[ranking ≥ BLOCKSIZE]
Truncate Ranking
Update Ranking Vector
Increment Number of Ranks
[Number of Rcv Ranks ≥ 1]
[else]
[ORDERING_VERSION=11]
Order Bundles (version 11)
[else]
[ORDERING_VERSION=12]
Order Bundles (version 12)
[else]
[ORDERING_VERSION=13]
Order Bundles (version 13)
[else]

Figure 5.5: MatlabEmulator ranking announcements flowchart

node only sends information if bundles resulted from the previous procedures, and while it has available bandwidth. The limitation of bandwidth is performed to emulate the physical channel and it is achieved by limiting the maximum number of bundles (or coded bundles) that the sender node can broadcast (`MAX_BUNDLES_SECOND`) in a time interval of `Ta` seconds. The broadcast transmission of information is emulated by crossing all the neighbors demonstrating the interest to send data to each of them. Only valid neighbors can received the information, whereby the intention of sending is limited to neighbors which link's RSSI is above the `RSSI_-MIN`. Another restriction is related to the `REPLICAS` macro. If it is active, both OBUs and RSUs can send bundles; otherwise, only RSUs can perform this operation.

5.2.7.3 Reception Process

At the end of the transmission process, the emulator knows which bundles (or coded bundles) a node wants to send. However, if the bandwidth is limited, not all of the intended bundles will be delivered to the nodes. On the other hand, if no reception bandwidth limitation is imposed, all the bundles are stored by their destination nodes.

As described in Figure 5.6, when the bandwidth limitation is active (`RX_FILTER` enable), the reception process starts with the assignment of bundles to the destination nodes (only OBUs). In this initial phase, the nodes (OBUs and RSUs) that each reception node (only

OBUs) will accept to communicate with, and how many bundles each will send through that node are determined. It is important to remember that each node is only able to receive a maximum of `MAX_BUNDLES_SECOND` bundles within a time interval of `Ta`. Thus, each receiver node will sort its neighbors (that it wants to send bundles to) according to the link's connection RSSI.

After that, if the number of nodes that want to communicate with the receiver node is lower or equal than `MAX_BUNDLES_SECOND`, these nodes are inspected in a circular way, and for each iteration the number of bundles that the sender node wants to transmit to the receiver node is assigned. On the other hand, if the number of sender nodes is equal or higher than `MAX_BUNDLES_SECOND`, one bundle of each sender node is assigned. These processes continue until the maximum number of bundles is selected, or all the bundles were assigned.

Once the assignment process ends, a set of local, global, and output variables are updated. The most important variables updated in this stage are the `bundles_stored` (bundles availability and number of hops), and `codbundles_stored` matrices (only coded bundles availability). An important note is the fact that, in order to update the number of hops when a node already had this specific bundle, it is always considered the bundle with a higher number of hops. So, if bundle X with n_1 hops is received and it is already in storage but it has n_2 hops, the field is updated with n_2 if $n_2 > n_1$.

5.2.7.4 Update Nodes State

This stage is particularly important when the LRGF strategy is selected. A node could decode a block of a file if at least `BLOCKSIZE` coded bundles of the same generation were received. Thus, if a node has at least `BLOCKSIZE` coded bundles of the same generation, it will decode the correspondent block. Therefore, all bundles associated with that block must be set to `1` in `bundles_stored` matrix, indicating their availability to be forwarded. This update is only done if the previous value was not `2` (meaning that they are no longer available but they were in the past). Moreover, since the receiver node has at least `BLOCKSIZE` coded bundles associated to the same block/generation, and regarding the considerations about the coding process in section 4.3, the receiver node is able to encode a complete generation of coded bundles. Thus, a set of additional coded bundles (equal to `GENSIZE-BLOCKSIZE`) are now available to be broadcasted, whereby the `codbundles_stored` matrix is updated.

Furthermore, in order to allow the determination of the time at which a specific node received the content, the `bundles_pernode` stores the current timestamp in the bundles received during this iteration.

5.2.8 Phase 3 - Statistical Analysis

Once all output variables are collected, they can be used to evaluate a set of metrics and provide a statistical analysis of the selected content distribution scheme. The resulting metrics are described as following.

1. Percentage of the file(s) stored in each node (only OBUs) past a specific time (default is 4 hours) since the beginning of the dissemination process.

2. E2E delay is the elapsed time to receive the complete file in an OBU since the beginning of the dissemination process. In the plot only the nodes that received all the bundles are considered.

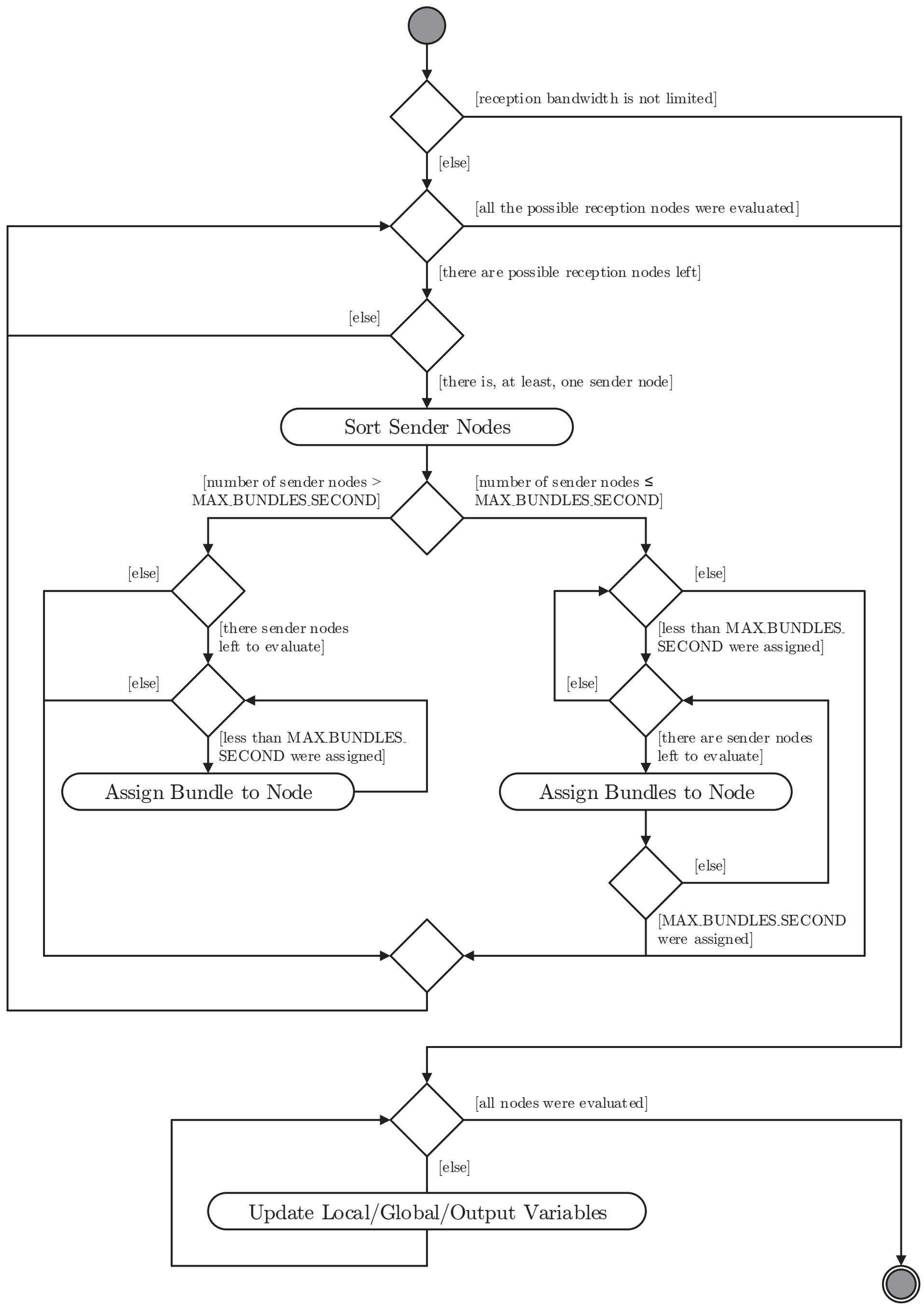

[reception bandwidth is not limited]
[else]
[all the possible reception nodes were evaluated]
[there are possible reception nodes left]
[else]
[there is, at least, one sender node]
Sort Sender Nodes
[number of sender nodes > MAX_BUNDLES_SECOND]
[number of sender nodes ≤ MAX_BUNDLES_SECOND]
[else]
[there sender nodes left to evaluate]
[else]
[less than MAX_BUNDLES_SECOND were assigned]
Assign Bundle to Node
[else]
[less than MAX_BUNDLES_SECOND were assigned]
[else]
[there are sender nodes left to evaluate]
Assign Bundles to Node
[else]
[MAX_BUNDLES_SECOND were assigned]
[all nodes were evaluated]
[else]
Update Local/Global/Output Variables

Figure 5.6: MatlabEmulator reception process flowchart

3. Delivery rate is considered to be the percentage of emulated OBUs that successfully downloaded the content under dissemination, and this metric is evaluated on an hour-by-hour basis.

4. Cumulative percentage of files distributed in the network (all the OBUs) along the experience.

5. Progress rate measures the rate of progress of completed download files averaged over all nodes over a specific time interval (default is 20 minutes).

6. Number of bundles received at time interval (`Ta`) and the cumulative number at the end of the experiment. The bundles are classified as bad (if a node already had the bundle), or good (if it did not).

7. Number of bundles rejected by bandwidth limitation at time interval (`Ta`) and the cumulative number at the end of experience. The rejected bundles are classified as bad (if a node did not have the bundle), or good (if it already had it).

8. Number of heard bundles by the network at time interval (`Ta`). This metric is evaluated through the cumulative sum of all the bundles that the OBUs heard in the medium, and it can be interpreted as a medium congestion metric.

9. Number and mean number of transmitted bundles by the network (OBUs and RSUs) at time interval (`Ta`).

10. Histogram counting the number of bundles that have a specific number of hops.

11. Histogram counting the number of OBUs that have a specific bundle.

5.2.9 Auxiliary Functions and Modules

In order to implement some strategies to disseminate information (to be explored in this document or in future approaches), two auxiliary functions were developed and described as follows.

The first one is created to generate the probability functions illustrated and defined in Figures 4.20, 4.21, and 4.23. The function returns a probability value (between 0 and 1) which can be used to define the probability of transmission. To evaluate this probability a set of four arguments are passed: X_1, X_2, P_1, and P_2. Their meaning varies according to the objective of the function used, as explored in section 4.4. Through the correct selection of the input arguments, it is possible to replicate the probability functions previously illustrated in section 4.4. The implemented algorithm is described as follows.

Algorithm 5.1 Evaluate probability according to a given probability function profile

```
Input: $0 < P_1, P_2 \leq 1 \wedge X_2 > X_1 \geq 0 \wedge x \geq 0$
Output: $y = f(X_1, X_2, P_1, P_2, x)$
  if $x \leq X_1$ then
    $y \Leftarrow P_1$
  else if $x \geq X_2$ then
    $y \Leftarrow P_2$
  else
    $m \Leftarrow (P_2 - P_1)/(X_2 - X_1)$
    $b \Leftarrow P_1 - m \times X_1$
    $y \Leftarrow m \times x + b$
  end if
```

Another interesting metric to be evaluated is the distance between two different nodes. In the proposed solutions there are no approaches using this criteria to perform the forwarding decision, although, in the future it could be useful. In addition to the transmission purpose, this parameter can be used to study the network behavior, or as in this document, to evaluate if a node enters a specific geographical area.

The developed function was adapted from a previous work of Ramin Shamshiri from the University of Florida, and evaluates the distance between two points according to their GPS coordinates. To perform this operation he uses the Carlson [156] model. The implemented algorithm to perform this calculation is illustrated in 5.2

Algorithm 5.2 Evaluate the distance between two points through the GPS coordinates

```
Output: $d = f(lat_1, long_1, lat_2, long_2)$
  $maj_{const} \Leftarrow 6378137$
  $min_{const} \Leftarrow 6356752.3142$
  $h \Leftarrow 334.9$
  $angle_1 \Leftarrow atan(min^2_{const}/maj^2_{const} \times tan(lat_1 \times \pi/180))) \times 180/\pi$
  $angle_2 \Leftarrow atan(min^2_{const}/maj^2_{const} \times tan(lat_2 \times \pi/180))) \times 180/\pi$
  $r_1 \Leftarrow \sqrt{1/(cos^2(angle_1 \times \pi/180)/maj^2_{const} + sin^2(angle_1 \times \pi/180)/min^2_{const})} + h$
  $r_2 \Leftarrow \sqrt{1/(cos^2(angle_2 \times \pi/180)/maj^2_{const} + sin^2(angle_2 \times \pi/180)/min^2_{const})} + h$
  $xy_1 \Leftarrow r_1 \times cos(angle_1 \times \pi/180)$
  $xy_2 \Leftarrow r_2 \times cos(angle_2 \times \pi/180)$
  $xy_3 \Leftarrow r_1 \times sin(angle_1 \times \pi/180)$
  $xy_4 \Leftarrow r_2 \times sin(angle_2 \times \pi/180)$
  $X \Leftarrow \sqrt{(xy_1 - xy_2)^2 + (xy_3 - xy_4)^2}$
  $Y \Leftarrow 2\pi \times ((xy_1 + xy_2)/2)/360 \times (long_1 - long_2)$
  $d \Leftarrow \sqrt{(X^2 + Y^2)}$
```

5.3 HelixEmulator

5.3.1 Introduction

The HelixEmulator is the second platform used to implement and evaluate the content distribution strategies previously proposed. This emulator was developed by in our research group (NAP) and aims to create a platform to develop and evaluate code specifically designed for DTN mechanisms (see subsection 3.5.2). It allows the creation of multiple processes that run DTN software (each one of them emulates a single node) providing a scalable framework to evaluate content distribution strategies or other routing algorithms. In this subsection an overview of the emulator structure is given along with several improvements that were developed and implemented during this dissertation.

5.3.2 Modifications and Improvements

Although the work so far developed in the emulator was quite significant, there are some improvements to be implemented along with new features. The list of modifications and improvements can be summarized as follows.

Update MySQL Database

Contrarily to other emulators (or simulators), in the HelixEmulator there is no mobility model. The mobility of the vehicles is introduced based on data acquired in the real vehicular network in Oporto city. As previously mentioned there is no log data with origin in RSUs whereby a backtracking approach was used to create it. As an example, if OBU A has as neighbor RSU B with a link's RSSI of X dBm, then RSU B also has a neighbor A with a link's RSSI of X dBm. The initial version of the HelixEmulator only had in its database information about OBUs, whereby it is necessary to update the MySQL database with information related to RSUs.

This information is generated in the same way as in MATLAB, and the generated matrices are saved as CSV files. After that, a Python script [157] was created to load this new information into the already existent tables in the Structured Query Language (SQL) database. As will be explained further in this document, there are two different datasets of log information which have different fields, and thus, two scripts were created.

Enable Broadcast Messages

The initial version of the HelixEmulator only provided unicast transmission of information. Thus, a significant set of changes was performed in order to provide the capability of communication in a broadcast manner. The emulator uses Zero-M Queue (ZMQ) [158] messages to communicate between processes (each process emulates a vehicle/node) in a server-subscribers model. Thus, it was necessary to create a ZMQ message and send it to multiple subscribers, maintaining a stable context and content during this procedure. This procedure was previously done through the creation of multiple messages (each one for each suitable subscriber) which was an incorrect approach.

Bandwitdh Limitation Module

An aspect neglected by the emulator was the bandwidth whereby no bandwidth, limitation was possible. Regarding this, a new module to limit the available bandwidth is implemented. A more detailed explanation is given in subsection 5.3.5.

Neighbor List Update Bug

The HelixEmulator updates the neighbor list of each node through control messages that carry information collected from the database which establish the ID of each neighbor and the link's RSSI. When a process (node) receives these messages, the neighbor list is updated. These messages are exchanged with a certain periodicity. Thus, if a node does not receive any message during this period, its neighbor list should be cleaned. However, this procedure was not working whereby this bug was fixed by checking if in the period any control message was received. If not, the neighbor list is empty.

Segment Nodes to Emulate

The initial version of the HelixEmulator emulates all nodes which have information within the database. When not all nodes are running (number of processes launched is lower than number of nodes in the database), the neighbors list was wrong. As an example, if node A is running, node B is not being emulated, and the original neighbor list of A is B and C, the list does not change and node B remains as neighbor of A (which is incorrect).

Thus, a different approach was created in order to correct this problem. Before starting HelixEmulator, two auxiliary files called *obu_list_filter_ref* and *rsu_list_filter_ref* must be created to specify which are the nodes to be emulated. Once created, the file is loaded to an internal structure that is used to guarantee that only information about the nodes running is loaded from the database into the emulator.

Modular Code Structure

The HelixEmulator source code project can be compiled to use the emulator or to generate code to be executed in the OBUs (this code needs to be cross-compiled before it runs on the boards). There are several macros (related with the implemented content distribution strategies) and modules that must be changed according to the platform (emulator or boards) where the Helix code needs to be executed. As such, the code was structured in a modular way in order to allow an easy compilation. As an example, when the code is compiled without the emulator (to be executed on the boards or directly on a machine such as a personal computer), the bandwidth module is not compiled and is excluded from the source code, since this limitation must be guaranteed by the physical medium.

Improve Statistical Analysis

The HelixEmulator did not have a robust platform to register metadata to be used in a further statistical procedure. The Logging module (see subsubsection 5.4.2.4) created for Helix software provides a way to perform statistical analysis of the emulator behavior along with the content distribution strategy efficiency.

5.3.3 Architecture and Operation Overview

The HelixEmulator was initially designed to be used on a single machine, running one emulator process and several Helix processes. However, it has been thought out to be scalable so that in the future one can run a distributed system to emulate the network. Therefore, the ZMQ [158] message queuing framework was used. Currently, it uses IPC for communications, but in the future TCP or UDP can be also used.

In the source code of the Helix software, it was possible to change the Socket module and make it listen to this IPC socket instead of an UDP socket (that is used on the OBUs). However, different types of messages flow between the emulator and the nodes (Helix processes), such as control messages sent by the emulator or regular Helix packets sent from node to node, and therefore a middle layer was introduced.

When Helix is being configured, a socket for IPC communications (to send to other nodes and receive control messages from the emulator) is created. Additionally, an in-proc socket was created to send regular packets to the Socket module. Thus, when a Helix process receives a control message (with node information), the important fields (such as the node's position and the neighboring list) are updated and the Socket block does not receive it since it is not a regular packet. However, when the Helix receives a regular packet (which is not a control message) it forwards it through the in-proc socket up to the Socket module, hence making Helix "believe" it has received a packet from another node.

In order to support the exchange of messages between the processes involved in the emulation procedure, an additional class was developed (see Figure 5.7).

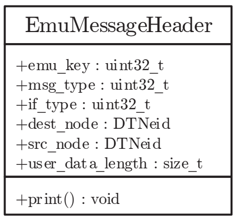


Figure 5.7: HelixEmulator message header class

This class allows the implementation of the previously described procedure, whereby when a node sends information to the other(s) node(s) an object of this class is created and its fields are initialized. The created object is a header joined with the Helix packet to be transmitted which is used by the HelixEmulator to route the packet to its destination.

5.3.4 Collection of Log Data from MySQL Database

As explained before, control messages are periodically sent from the emulator to all the nodes to update their internal state. This information is real information collected on a testbed and is currently being queried from a MySQL database. As shown in Figure 5.8 three tables were created: "timestamps", "neighbors" and "rsu".

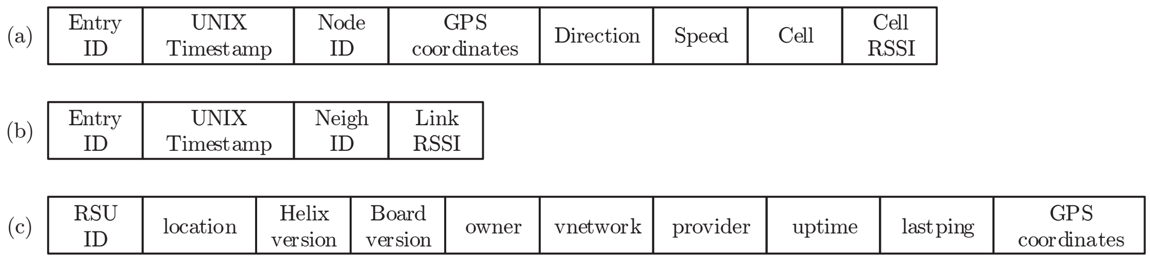


Figure 5.8: HelixEmulator database tables: (a) timestamps, (b) neighbors, and (c) rsu

The "timestamps" table contains information about a timestamp regarding a specific node ID. The second table relates a row in the "timestamps" table to a set of neighbors that was seen by the corresponding node ID at that sample. Finally, the "rsu" table contains information regarding RSUs at the time the data collection was performed.

Once the files that contain the 24h data collection are obtained, their contents should be imported to a database in order to be used in the emulator. A Python [157] script was created to help on this task. However, before using it, it is crucial to check if the data log files still are in agreement with the implemented Python scripts. Since the data log files of the second set of collection data were different from the initial dataset, several modifications were performed on the provided script to accommodate the changes.

The emulator retrieves the information from the MySQL database in the JSON format. This is done using an Apache server and Hypertext Preprocessor (PHP) with connectors to the MySQL database. There are two APIs available: one to get the list of RSUs and another one to get the network status at a given timestamp. Once the server is up and running and both APIs are accessible, two simple HTTP GET requests can be made to retrieve the stored information in the database. After the successful queries, the HelixEmulator parses the JSON into a well-known and defined emulator control packet to send to every node in order for them to update their internal state.

This control message is implemented using a class with the structure illustrated in Figure 5.9. It contains several information very important to the emulator, such as the position of nodes, list of neighbors and their links characteristics.

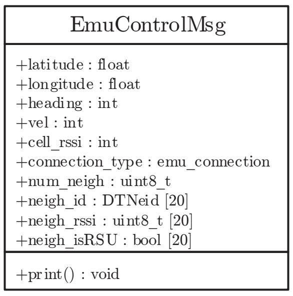


Figure 5.9: HelixEmulator control message class

5.3.5 Bandwidth Limiter Module

The HelixEmulator did not have any mechanism for bandwidth limitation, whereby it did not approach the physical channel or service's bandwidth characteristics. Thus, an additional module was developed to be included in the HelixEmulator.

The Bandwidth Limiter module is responsible for imposing a maximum reception and transmission bandwidth in each node. The class that implements it is illustrated in Figure 5.10. The fundamental data members are two counters named of `rx_packets_counter` and `tx_packets_counter`, which respectively count the number of packets received and transmitted by a node. These counters are updated every time that a node receives or transmits a packet using `incRx` and `incTx` methods. Furthermore, internally two macros are defined, called `MAX_RX_PACKETS_PER_PERIOD` and `MAX_TX_PACKETS_PER_PERIOD`, that respectively define the maximum number of packets which a node can receive and transmit in a sample period (macro called `BANDWIDTH_SAMPLE_PERIOD`). All the access and manipulation of variables are performed using thread-safe methods.

This module has a dedicated thread (`runBandwidthLim`) which is responsible for periodically (with a period of `BANDWIDTH_SAMPLE_PERIOD`) reseting the internal counters. Thus, every time that a node wants to send a packet or is receiving a packet from other nodes, it must check its internal bandwidth counters to ensure the availability of bandwidth (see Figure 5.11). Such queries are performed using the `canRX` and `canTx` methods.

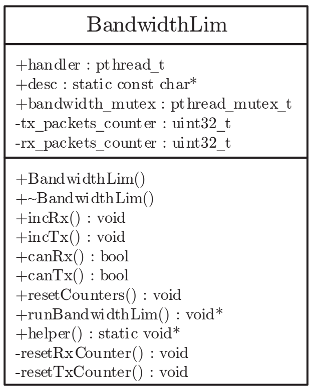


Figure 5.10: HelixEmulator BandwidthLim module collaboration diagram

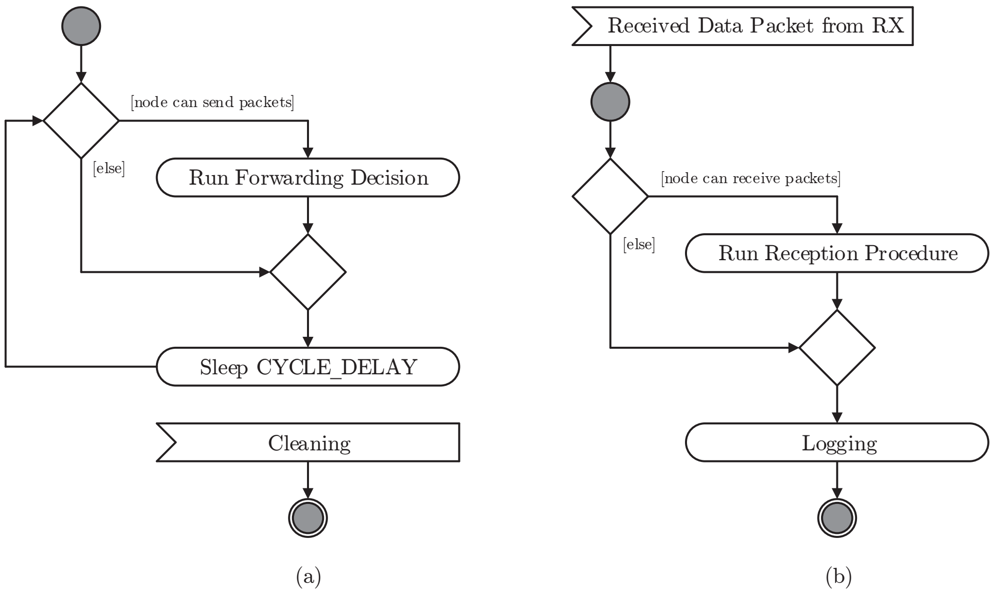


Figure 5.11: HelixEmulator BandwidthLim (a) forwarding decision and (b) reception flowcharts

5.4 Helix Integration

5.4.1 Introduction

In order to implement the proposed content distribution strategies in the Helix software several steps need to be taken and challenges to be overcome. Thus, the Helix integration stage is the crucial phase of this work. This section describes the modifications regarding the implementation of the content distribution schemes in the Helix software and the enforcement of the transmission decision capabilities.

5.4.2 Global Modifications

Several modifications to the Helix architecture were performed in order to implement the content distribution schemes presented before. As illustrated in Figure 5.12, the initial architecture of Helix (see Figure 3.10) is modified mainly in the Routing, Storage, RX modules, and packet header. Furthermore, two auxiliary modules were added, **Handler** (which has a set of sub-modules) and **Logging**. The Handler module can be divided into three sub-modules according to the selected content distribution scheme: HandlerLNHF, HandlerLRBF, or HandlerLRGF. The Random strategy is directly implemented in the peeking procedure of the Storage module. The Logging module is introduced to collect log data from content distribution experiments and, at the end, perform a statistical analysis.

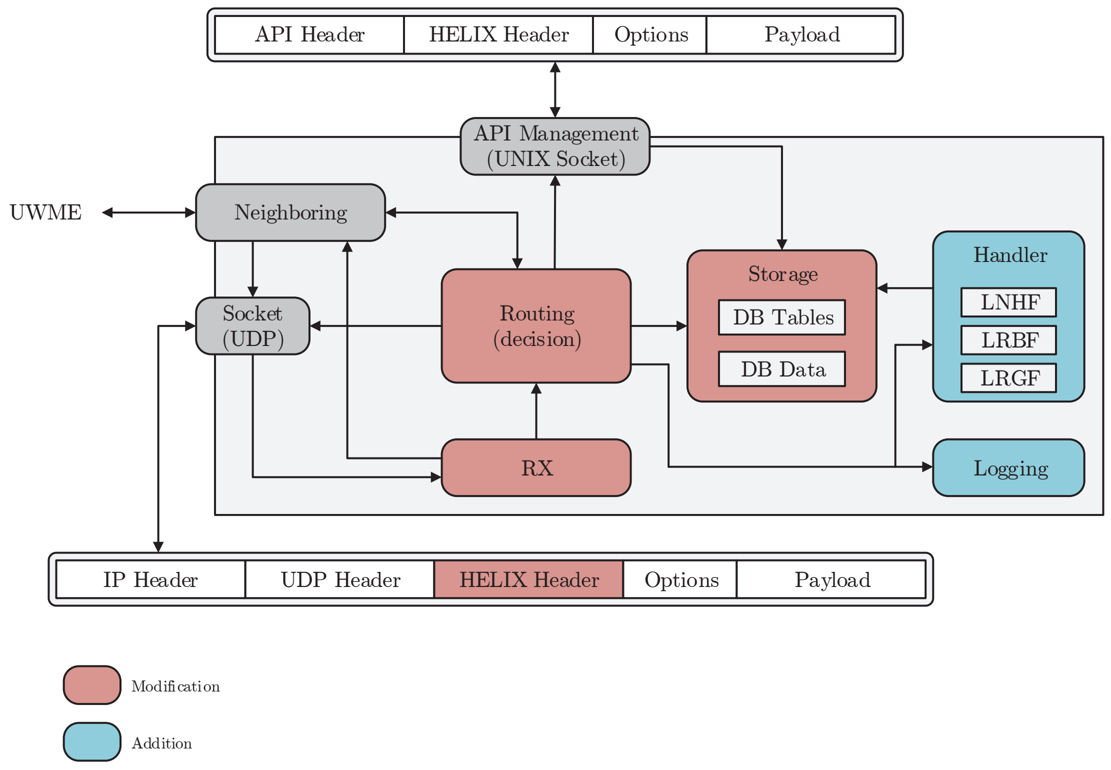


Figure 5.12: Helix architecture - modified and added modules

In the following sections, the global modifications performed to integrate the selected content distribution schemes into Helix software are described.

5.4.2.1 Helix Support Library and Packet Handling

Along with the Helix daemon, a support library, called libhelix, was developed with several functions to be used by applications and to define a few general specifications. Our focus is not related with helix applications, whereby the auxiliary functions developed for that will not be described. The key functionality introduced by this support library to our work is the structure of Helix packet header as described in Figure 3.10.

In order to integrate the content dissemination strategies into Helix software, the Helix-Header was modified. As illustrated in 5.13, several fields were modified or introduced. The following list describes their (new) meaning:

- *serviceID*: this field identifies the service to which this packet belongs. As new types of services were introduced, this field was changed.
- *nNeigh*: the initial definition of this file was the current number of neighbors that received the packet or, in the case of neighboring announcements, it represented the total number of neighbors of a certain node. However, in the implemented strategies it has a slightly different meaning, being responsible to identify the number of hops that this packet has traveled.
- *flags*: new types of packets related with new service IDs (and corresponding content dissemination schemes) were introduced, namely the advertisements. Thus, this field can have a different meaning in an advertisement packet.
- *fileID*: identifies the file to which this packet belongs.
- *totalPacketsFile*: identifies the total number of packets which the file identified by fileID has.
- *blockID*: identifies the block to which this packet belongs (only relevant for LRGF strategy).
- *blockSize*: identifies the size of a block in number of packets (only relevant for LRGF strategy).
- *genSize*: identifies the size of a generation in number of coded packets (only relevant for LRGF strategy).

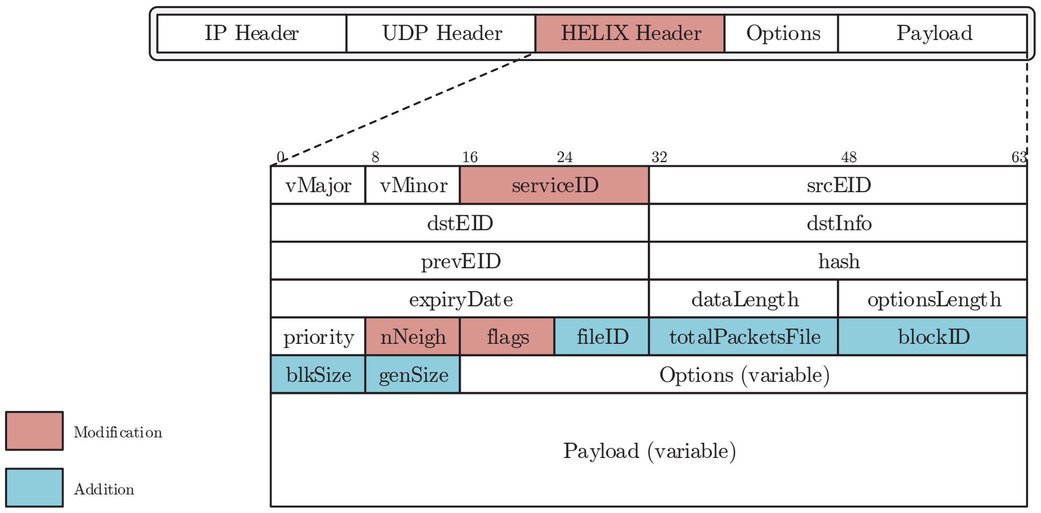


Figure 5.13: Helix packet header - modified and added fields

Four new service IDs (`DTN_CONTENT_DIST_1...4`) were allocated to the content distribution service. These multiple IDs can be used to identify different types of contents or specific services within the content distribution (e.g. high priority, only for a specific set of nodes, etc.). In addition to that, a new type of packet was introduced, the advertisement (used in LRBF and LRGF strategies). Thus, a new mask to identify the type of packet through the flag field was added and called `DTN_FLAG_ADV_MASK`.

The Helix developers created an auxiliary class to handle Helix packets. Within this class, several methods were created to, for example, make a new packet, print it, serialize (or deserialize) an outcoming (or incoming) packet, among others. Since the Helix header was modified and new fields were added, some of the previous methods had to be rebuilt. Thus, the serialize and deserialize methods were modified to introduce the serialization (or deserialization) of the new added fields in these processes.

5.4.2.2 RX Module

As mentioned in section 3.5.2.2, Helix has a module responsible for forwarding the received packets from Socket module to other modules (Routing or Neighboring). This classification is based on the service ID and/or in flags presented in the received packet. Since a new type of flag was introduced to identify advertisement packets, the RX module was modified to deal with it. Thus, as illustrated in Figure 5.14, a new action was introduced comparatively to the initial operation flow. If the packet is an advertisement, its handling is performed by the Routing module using the `advHandler` method.

As an addition, Figure 5.14 also illustrates the remaining handling procedures when a packet is received. If the packet is a neighboring announcement, it will be forwarded to the Neighboring module to update information related with the node's vicinity. All the other types of packet acknowledgments, advertisements, and data, are forwarded to the Routing module and being handled by `ackHandler`, `advHandler`, and `pktHandler`, respectively.

5.4.2.3 Storage Module

As mentioned before (see section 3.5.2.2), the storage has the responsibility of persistently keep the data and control information, but also to provide methods to access and retrieve such information. Thus, several peeking methods were developed by the authors, and it is possible to peek a packet according to its destination, serviceID, or expiration time. However, for the implemented content distribution schemes, new methods of peeking were needed according to other parameters, such as packet hash which is used to identify the packets in the Handler modules' internal structure. Therefore, two new peeking methods were created:

- `peek_hash(uint32_t packethash)`: this function returns a copy of the stored packet identified by `packetHash`.
- `peek_rand()`: this function returns a copy of the stored packet in a random way, so it returns any other packet mapped in storage and available to be returned.

Both methods only peek packets from the expiry table (list of available packets to be returned).

The packets to be forwarded are organized by storage module in two tables: *Expiry* and *onHold*. In the first one the packets are sorted in ascending order according with their lifetime. When a packet is peeked from the expiry table, it could be transfered to the onHold table for a certain period of time from `ST_MIN_ONHOLD` to `ST_MAX_ONHOLD`. This feature was introduced to prevent the starvation of packets with a higher lifetime, since no intelligence was introduced in the peeking procedure by expiry time, and thus the packet with the lowest lifetime was always peeked first. However, with the implementation of an intelligent choice of packets

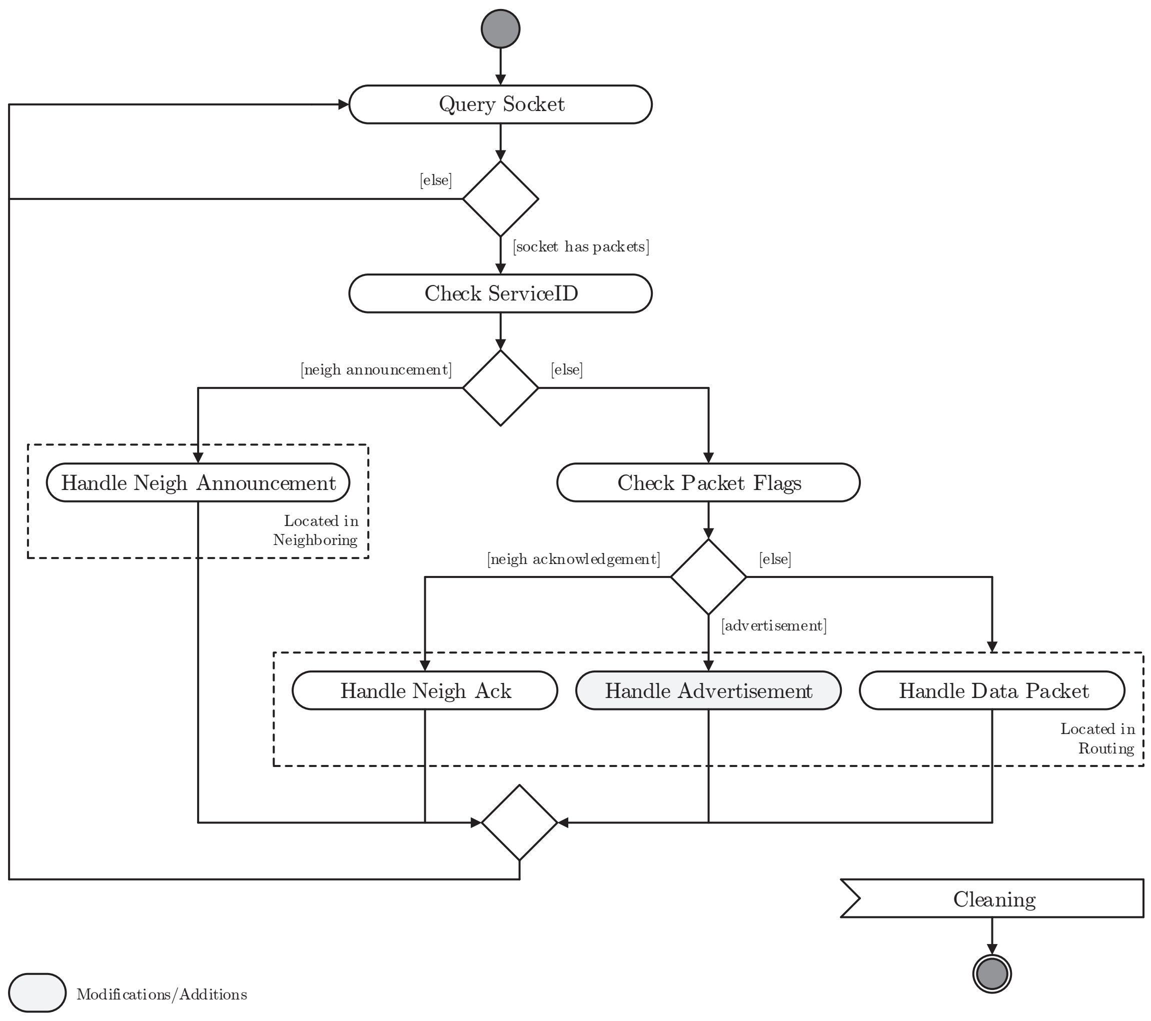


Figure 5.14: Helix reception process flowchart

(according to packet hash) there is no need to perform such transfer of packets between the two tables since the starvation problem, caused by the continuously peeking of the packet with the lower expiry date in the expiry table, is avoided. Thus, whereby the `ST_MIN_ONHOLD` to `ST_MAX_ONHOLD` macros are set to `0`.

Finally, the mapping of data in the hard disk is done using an approach similar to a memory mapping scheme [159]. Within its implementation, the maximum number of pages allowed to be used, `MAX_PAGES` is defined. This is a key macro and must be considered during the implementation of content distribution schemes since a lower value can not be enough to map all the files to be disseminated if it has a high size. A value of `300` was selected, instead of the default value of `100`, since it proved enough to map a file of approximately 200 MB.

5.4.2.4 Logging Module

The main factor which leads to the implementation of this module from the start was the lacking of features and functionalities related with content distribution evaluation within

Helix software. Furthermore, and following a similar approach that was used in the MatlabEmulator, it is crucial to collect log information during the dissemination process that can be used at the end to perform a statistical analysis of the procedure. The class used to implement this module is described in Figure 5.15.

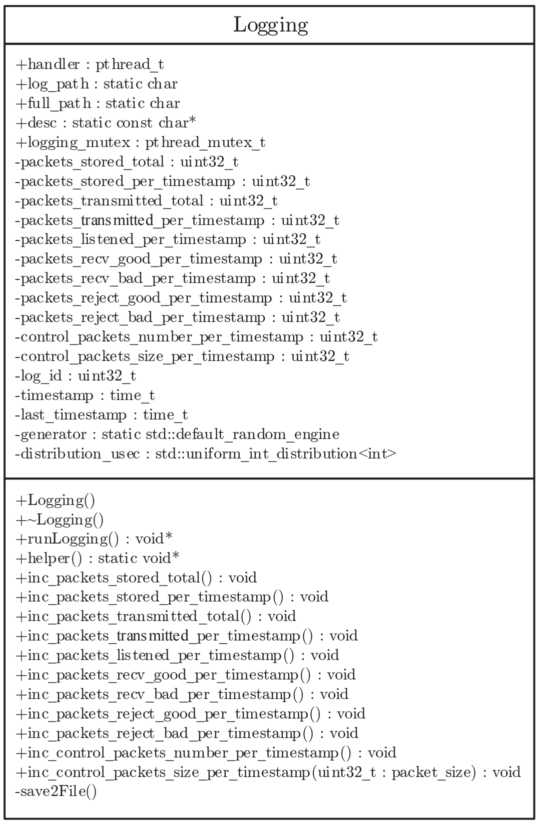

Logging
+handler : pthread_t
+log_path : static char
+full_path : static char
+desc : static const char*
+logging_mutex : pthread_mutex_t
-packets_stored_total : uint32_t
-packets_stored_per_timestamp : uint32_t
-packets_transmitted_total : uint32_t
-packets_transmitted_per_timestamp : uint32_t
-packets_listened_per_timestamp : uint32_t
-packets_recv_good_per_timestamp : uint32_t
-packets_recv_bad_per_timestamp : uint32_t
-packets_reject_good_per_timestamp : uint32_t
-packets_reject_bad_per_timestamp : uint32_t
-control_packets_number_per_timestamp : uint32_t
-control_packets_size_per_timestamp : uint32_t
-log_id : uint32_t
-timestamp : time_t
-last_timestamp : time_t
-generator : static std::default_random_engine
-distribution_usec : std::uniform_int_distribution<int>
+Logging()
+~Logging()
+runLogging() : void*
+helper() : static void*
+inc_packets_stored_total() : void
+inc_packets_stored_per_timestamp() : void
+inc_packets_transmitted_total() : void
+inc_packets_transmitted_per_timestamp() : void
+inc_packets_listened_per_timestamp() : void
+inc_packets_recv_good_per_timestamp() : void
+inc_packets_recv_bad_per_timestamp() : void
+inc_packets_reject_good_per_timestamp() : void
+inc_packets_reject_bad_per_timestamp() : void
+inc_control_packets_number_per_timestamp() : void
+inc_control_packets_size_per_timestamp(uint32_t : packet_size) : void
-save2File()

Figure 5.15: Helix Logging module collaboration diagram

Thus, the Logging module performs a randomly time register of log data collected along the process. As illustrated in Figure 5.16, the module can operate in two different ways, according to its framework. If it is being used with the HelixEmulator, the timestamp is collected according to the emulation timestamp. On the other hand, if it is running on a dedicated board or computer, the collected timestamp is the actual time. In addition to this, if the HelixEmulator is active, a random sleep is added to randomize the access of the emulation timestamp among all the processes running concurrently. After that, and if the timestamp changes from the previous one, the log data is saved to a file to be further analyzed. At the end, the thread sleeps a random time between `MIN_TIME_STEP_LOG` and `MAX_TIME_STEP_LOG`. If the HelixEmulator is active, it is important that these macros are at equal distance from the emulation time step, since some log data has to be collected with a constant periodicity equal to the emulation time step.

Several information is collected along the dissemination process. A definition of the variables used to store this information is presented below.

- `log_id`: unique number which identifies the log entry.

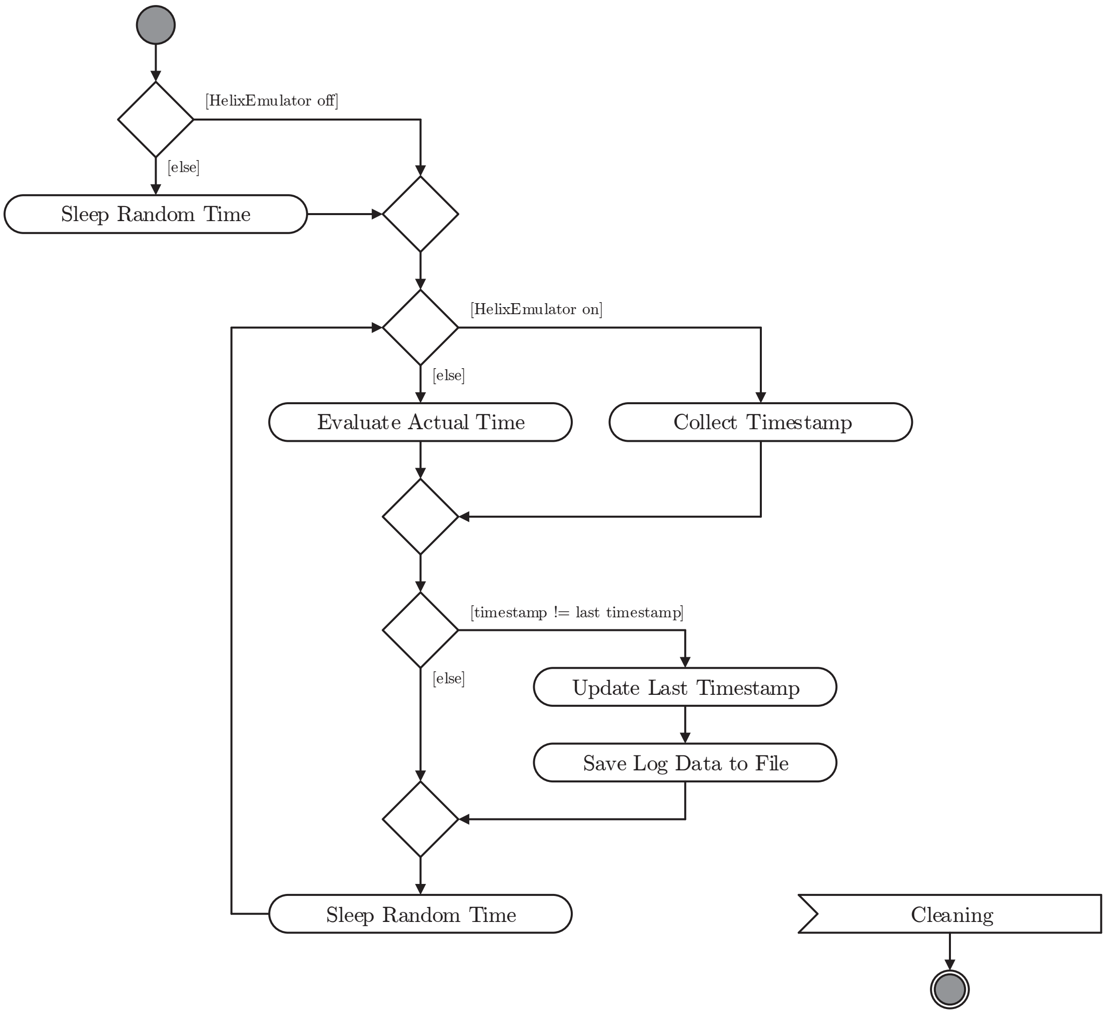


Figure 5.16: Logging module operation flowchart

- `node_eid`: endpoint identifier of the node where information was generated.
- `timestamp`: UNIX timestamp which identifies the time when an entry was registered.
- `packets_stored_total`: total number of files stored by a node since the beginning of the dissemination procedure.
- `packets_transmitted_total`: total number of files transmitted by a node since the beginning of the dissemination procedure.
- `packets_stored_per_timestamp`: number of packets stored during a sample period.
- `packets_transmitted_per_timestamp`: number of packets transmitted during a sample period.

- `packets_listened_per_timestamp`: number of packets heard in the wireless medium during a sample period.
- `packets_recv_good_per_timestamp`: number of packets stored by a node which it does not have yet during a sample period.
- `packets_recv_bad_per_timestamp`: number of packets stored by a node which it already has during a sample period.
- `control_packets_number_per_timestamp`: number of control packets transmitted during a sample period.
- `control_packets_size_per_timestamp`: cumulative size of control packets transmitted during a sample period.
- `expiry_table_size`: actual size of the expiry table of storage.
- `onhold_table_size`: actual size of the onHold table of storage.

After saving the log data into a file, all the variables related with a sample period are reset to zero. The resulting file is a CSV file which can be easily treated (e.g. in MATLAB) to perform statistical analysis.

All the access and manipulation of internal variables are performed using thread-safe methods since multiple threads may try to access to them simultaneously.

5.4.3 Routing Module Description

The integration procedures performed to implement the content distributions schemes previously presented rely mostly on modifications and additions to the Routing module of the Helix software. Thus, it is important to better understand how this module is implemented and how it works, whereby it will be presented in this section.

A simplified version of the Routing module collaboration diagram is presented in Figure 5.17. In this diagram the integrated modules and methods to handle the content distribution schemes are already included, together with the Logging and BandwidthLim (only used if HelixEmulator is running) modules previously discussed. To perform the forwarding decision of packets, the Routing module creates a set of objects, namely to access storage data and metadata, communication modules (Socket module), and possible APIs that are running. Furthermore, it is responsible for the integration of handling modules of content distribution strategies.

The Routing module is a core module of Helix since it is responsible for treating the received packets and decide what packets should be sent and to whom. Thus, it defines an interface to handle the incoming and departing packets. The most important methods to treat incoming packets from the RX module are the `ackHandler` which is responsible for acknowledgment packets, `pktHandler` for data packets, and `advHandler` (integrated for content distribution schemes implementation) to deal with advertisement packets. Regarding the interaction with the Socket module to send control packets, sending methods to send packets to other nodes are defined: `sendNeighAck`, `sendEndAck`, and `sendContAdv`. When it is necessary to send a data packet, the Routing module directly uses the pointer to the Socket object (`helixSocket`) to send it in broadcast or unicast way. Other methods related

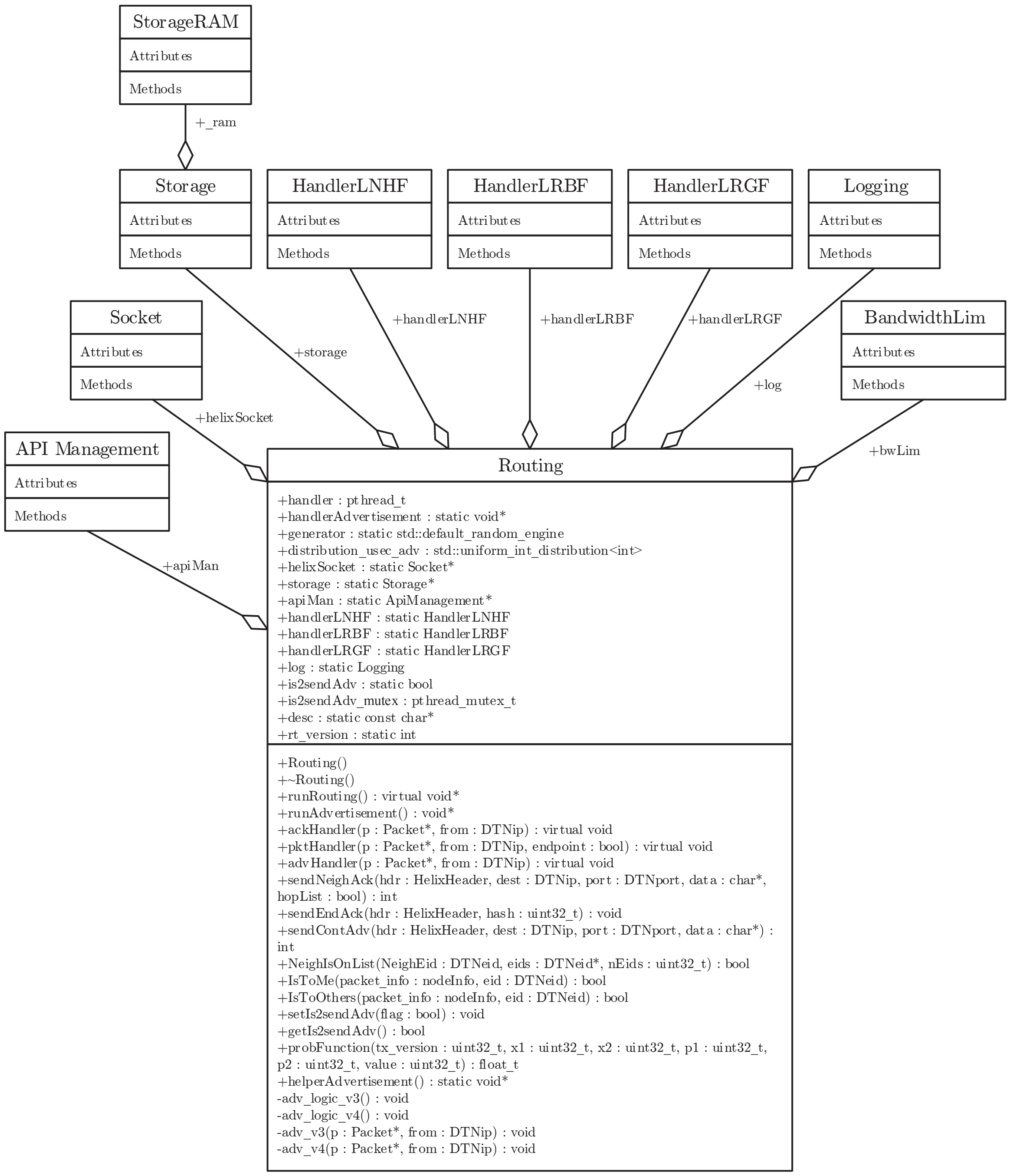

StorageRAM
Attributes
Methods
+_ram
Storage
Attributes
Methods
HandlerLNHF
Attributes
Methods
HandlerLRBF
Attributes
Methods
HandlerLRGF
Attributes
Methods
Logging
Attributes
Methods
Socket
Attributes
Methods
BandwidthLim
Attributes
Methods
API Management
Attributes
Methods
+storage
+handlerLNHF
+handlerLRBF
+handlerLRGF
+log
+helixSocket
+bwLim
+apiMan
Routing
+handler : pthread_t
+handlerAdvertisement : static void*
+generator : static std::default_random_engine
+distribution_usec_adv : std::uniform_int_distribution<int>
+helixSocket : static Socket*
+storage : static Storage*
+apiMan : static ApiManagement*
+handlerLNHF : static HandlerLNHF
+handlerLRBF : static HandlerLRBF
+handlerLRGF : static HandlerLRGF
+log : static Logging
+is2sendAdv : static bool
+is2sendAdv_mutex : pthread_mutex_t
+desc : static const char*
+rt_version : static int
+Routing()
+~Routing()
+runRouting() : virtual void*
+runAdvertisement() : void*
+ackHandler(p : Packet*, from : DTNip) : virtual void
+pktHandler(p : Packet*, from : DTNip, endpoint : bool) : virtual void
+advHandler(p : Packet*, from : DTNip) : virtual void
+sendNeighAck(hdr : HelixHeader, dest : DTNip, port : DTNport, data : char*, hopList : bool) : int
+sendEndAck(hdr : HelixHeader, hash : uint32_t) : void
+sendContAdv(hdr : HelixHeader, dest : DTNip, port : DTNport, data : char*) : int
+NeighIsOnList(NeighEid : DTNeid, eids : DTNeid*, nEids : uint32_t) : bool
+IsToMe(packet_info : nodeInfo, eid : DTNeid) : bool
+IsToOthers(packet_info : nodeInfo, eid : DTNeid) : bool
+setIs2sendAdv(flag : bool) : void
+getIs2sendAdv() : bool
+probFunction(tx_version : uint32_t, x1 : uint32_t, x2 : uint32_t, p1 : uint32_t, p2 : uint32_t, value : uint32_t) : float_t
+helperAdvertisement() : static void*
-adv_logic_v3() : void
-adv_logic_v4() : void
-adv_v3(p : Packet*, from : DTNip) : void
-adv_v4(p : Packet*, from : DTNip) : void

Figure 5.17: Helix Routing module collaboration diagram

specifically to content distribution strategies integration, like advertisement related methods, will be discussed further in this document.

As mentioned before, Helix adopted a hybrid routing solution since it routes per neighbor and per packet type. The first routing decision is based on the packet type (data or control), but the remaining process depends on the node type. Thus, it presents the inheritance diagram illustrated in Figure 5.18. In this diagram the structure of `RoutingServer` and `RoutingWiFi` classes is omitted since in this document it is assumed that only OBUs and RSUs are running as content distribution nodes. Those classes are used by Wi-Fi (e.g. sensors) and Server type of nodes as handlers of their routing decisions.

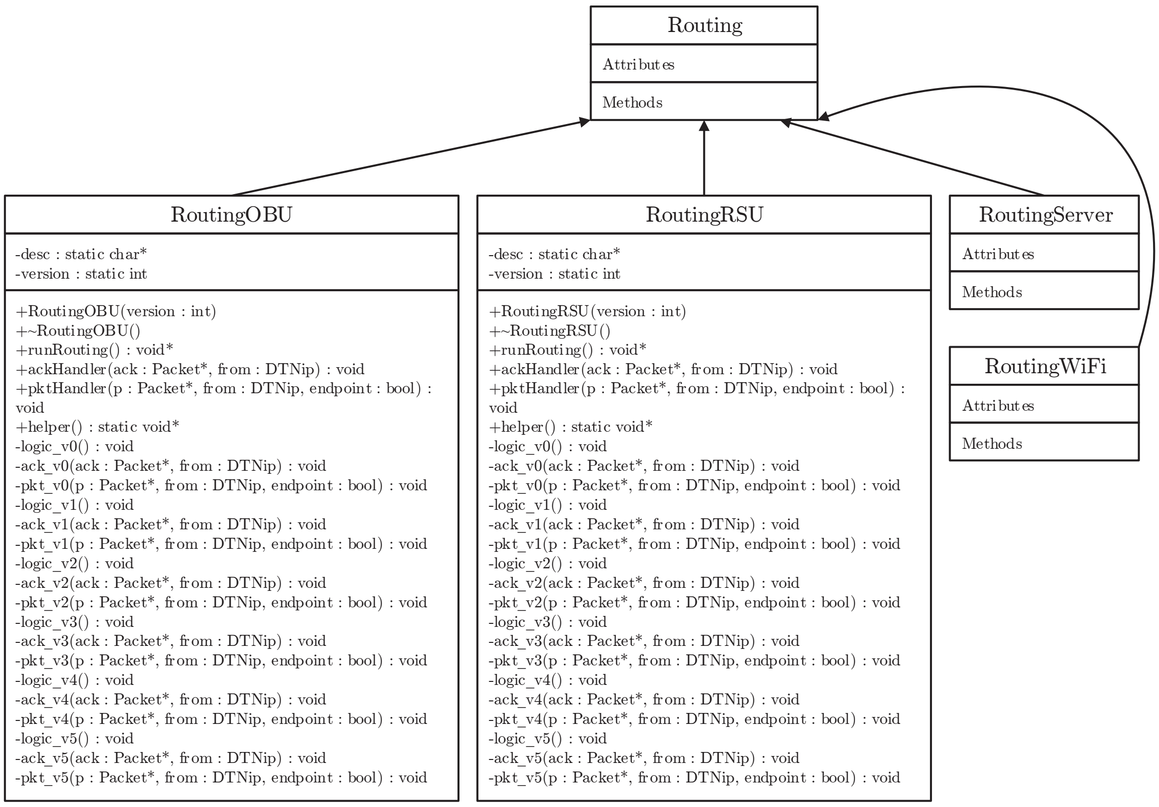

Routing
Attributes
Methods
RoutingOBU
-desc : static char*
-version : static int
+RoutingOBU(version : int)
+~RoutingOBU()
+runRouting() : void*
+ackHandler(ack : Packet*, from : DTNip) : void
+pktHandler(p : Packet*, from : DTNip, endpoint : bool) : void
+helper() : static void*
-logic_v0() : void
-ack_v0(ack : Packet*, from : DTNip) : void
-pkt_v0(p : Packet*, from : DTNip, endpoint : bool) : void
-logic_v1() : void
-ack_v1(ack : Packet*, from : DTNip) : void
-pkt_v1(p : Packet*, from : DTNip, endpoint : bool) : void
-logic_v2() : void
-ack_v2(ack : Packet*, from : DTNip) : void
-pkt_v2(p : Packet*, from : DTNip, endpoint : bool) : void
-logic_v3() : void
-ack_v3(ack : Packet*, from : DTNip) : void
-pkt_v3(p : Packet*, from : DTNip, endpoint : bool) : void
-logic_v4() : void
-ack_v4(ack : Packet*, from : DTNip) : void
-pkt_v4(p : Packet*, from : DTNip, endpoint : bool) : void
-logic_v5() : void
-ack_v5(ack : Packet*, from : DTNip) : void
-pkt_v5(p : Packet*, from : DTNip, endpoint : bool) : void
RoutingRSU
-desc : static char*
-version : static int
+RoutingRSU(version : int)
+~RoutingRSU()
+runRouting() : void*
+ackHandler(ack : Packet*, from : DTNip) : void
+pktHandler(p : Packet*, from : DTNip, endpoint : bool) : void
+helper() : static void*
-logic_v0() : void
-ack_v0(ack : Packet*, from : DTNip) : void
-pkt_v0(p : Packet*, from : DTNip, endpoint : bool) : void
-logic_v1() : void
-ack_v1(ack : Packet*, from : DTNip) : void
-pkt_v1(p : Packet*, from : DTNip, endpoint : bool) : void
-logic_v2() : void
-ack_v2(ack : Packet*, from : DTNip) : void
-pkt_v2(p : Packet*, from : DTNip, endpoint : bool) : void
-logic_v3() : void
-ack_v3(ack : Packet*, from : DTNip) : void
-pkt_v3(p : Packet*, from : DTNip, endpoint : bool) : void
-logic_v4() : void
-ack_v4(ack : Packet*, from : DTNip) : void
-pkt_v4(p : Packet*, from : DTNip, endpoint : bool) : void
-logic_v5() : void
-ack_v5(ack : Packet*, from : DTNip) : void
-pkt_v5(p : Packet*, from : DTNip, endpoint : bool) : void
RoutingServer
Attributes
Methods
RoutingWiFi
Attributes
Methods

Figure 5.18: Helix Routing module inheritance diagram

Both `RoutingOBU` and `RoutingRSU` classes are quite similar regarding their interface, although their implementation (content of methods) is very different since they have completely distinct behaviors when receiving or sending a packet. The implementation of each method belonging to versions 2 to 5 will be discussed further in this document. `RoutingOBU` and `RoutingRSU` are derived from Routing class which has defined virtual methods to treat packets (both sent and received). Thus, these methods are redefined by these two classes in order to implement their own routing decision according to their type. Included in these methods are `ackHandler` and `pktHandler`, which are invoked when a node receives, respectively an acknowledgment or data packet. Moreover, the method responsible for the forwarding decision (`runRouting`), which is invoked by a periodic thread to send packets, is redefined. Each one of the previous methods can be split into a set of auxiliary methods used according to the routing version which is running. Versions 2 to 5 are reserved for content distribution strategies (2-LNHF, 3-LRBF, 4-LRGF, and 5-Random).

5.4.4 Content Distribution Schemes Implementation

In the following topics the implementation and integration of the previously mentioned content distribution schemes, namely Random, LNHF, LRBF, and LRGF, is presented and discussed. The explanation includes a description of the handling class (when it was implemented), main auxiliary structures, and a description of the reception and forwarding data and, when they exist, advertisement packets and auxiliary threads.

Since the communication between nodes is always performed in a broadcasted manner, no acknowledgment packet was implemented, whereby there is no need to implement or redefine handling methods to this type of packets. Moreover, when the flowcharts and flows of information are described, it is assumed that Helix is running on a real board and not in HelixEmulator, whereby the BandwidthLim module is not included in this analysis. However, a topic about how to generate the initial content distribution packets in RSUs in order to initially store the content, since it is assumed that there is no local server in this procedure is included.

5.4.4.1 Random

The random strategy relies mostly on the Storage module manipulation, not requiring an implementation of any additional module or structures. This strategy relies on peeking a random packet from storage. Thus, a new method to randomly peek packets from storage was implemented. The `peek_rand` method was added to Storage module and returns a random packet from the expiry table if it contains any packets.

Forwarding Decision

The forwarding decision flowchart is illustrated in Figure 5.19, which describes the behavior of the `runRouting` thread when the Random strategy is selected. First the sender node checks if it has neighbors and if all of them are OBUs. If this condition fails the process jumps to its end. On the other hand, and if it has any packets in its storage, it proceeds to a random peek of a packet using the `peek_rand` method. If any packet was peeked and if it belongs to a content that is under dissemination, the field responsible for identifying the source node of the packet (`prev_EID`) is updated and the packet is sent in broadcast to its neighbors. If the previous conditions are not fulfilled, the process jumps to its end. Before finishing, the logging variables like `packets_transmitted_total` and `packets_transmitted_per_timestamp` are incremented. At the end, the process sleeps for `CYCLE_DELAY` microseconds (default value is 200 ms) and after that it resumes the process. The thread ends when a signal of cleaning is activated, jumping to the clean up functions (see Figure 3.14).

Reception of a Data Packet

In Figure 5.20 the procedure carried out when a data packet is received from the RX module is described. The analysis is divided in two processes, since the procedure varies according to the node type. If the receiver is an OBU and the packet was received from a WAVE interface, the node ID is added to the packet's hop list and, if the node does not have store this packet, it pushes the packet to storage. On the other hand, if one of the previous conditions fail, the procedure jumps to the logging phase. Thus, when the node does not have the received packet, a set of logging counters are updated, registering the number of stored packets in total and within this timestamp, and the number of good packets

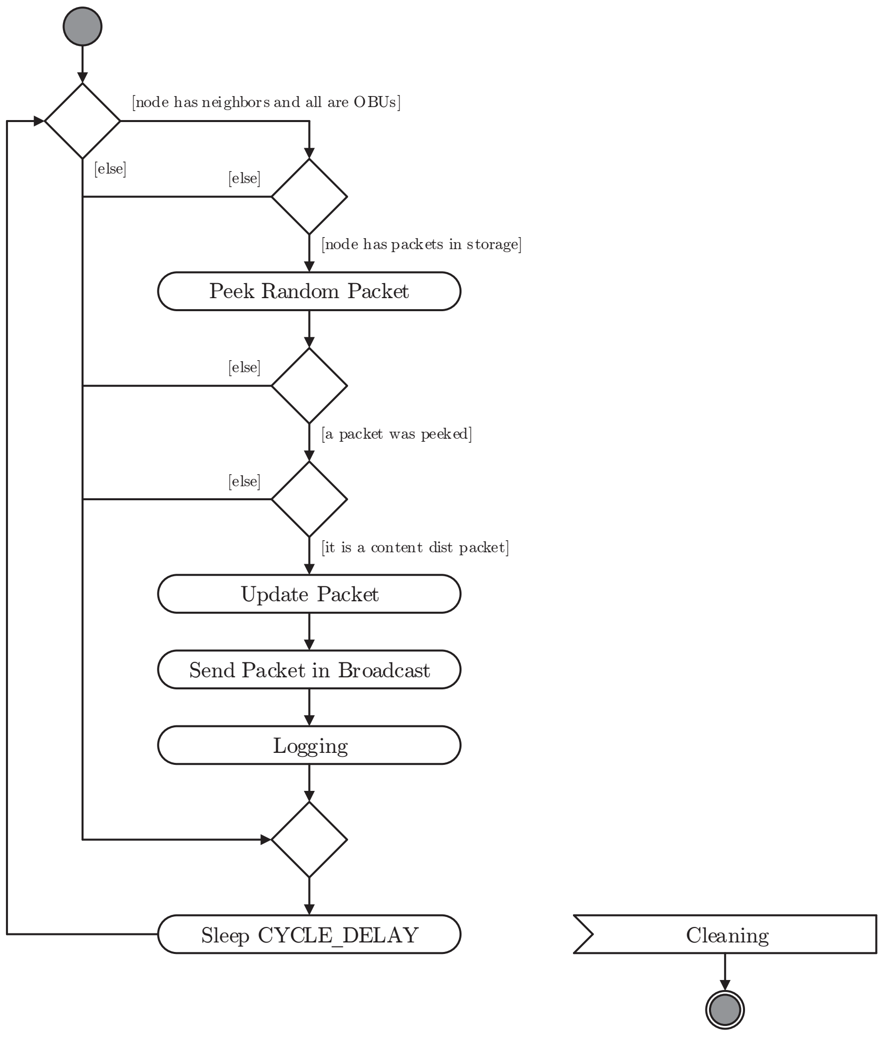
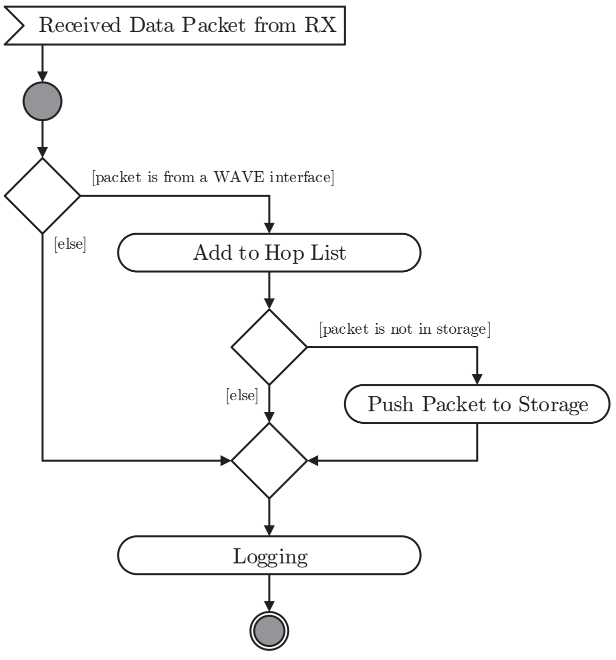


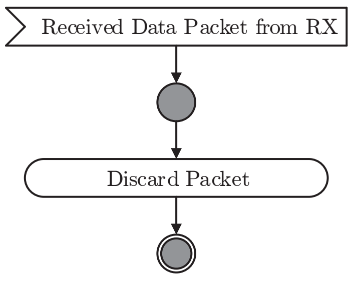
Figure 5.19: Helix integration Random strategy data packet forwarding decision flowchart

received (`packets_stored_total`, `packets_stored_per_timestamp`, and `packets_recv_-good_per_timestamp`). However, if a packet already stored was heard, the number of bad packets received is incremented (`packets_recv_bad_per_timestamp`). In both cases, the number of heard packets within this timestamp (`packets_listened_per_timestamp`) is also incremented. On the other hand, if the receiver node is an RSU, the packet is discarded since the dissemination procedure is always downstream (from RSUs to OBUs or among OBUs).

Initial Generation of Content Distribution Packets

The procedure to initially store packets in RSUs is illustrated in Figure 5.21. Once the number of packets is defined, all the packets are created and stored. First the packet's header is initialized and a packet is created (join of header and payload) and pushed to storage.


Received Data Packet from RX
[packet is from a WAVE interface]
[else]
Add to Hop List
[packet is not in storage]
[else]
Push Packet to Storage
Logging
(a)



(b)

Figure 5.20: Helix integration Random strategy data packet reception in (a) an OBU and (b) an RSU flowcharts

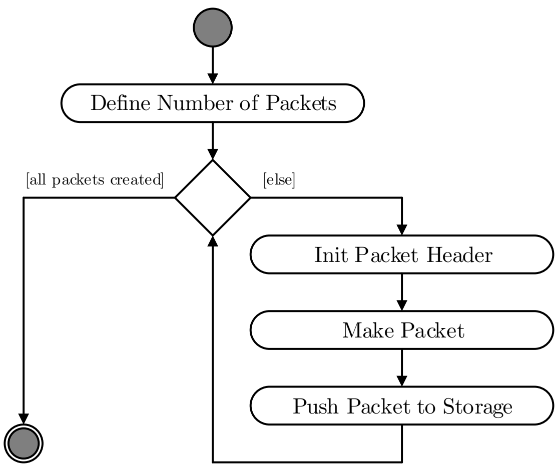


Figure 5.21: Helix integration Random strategy generation of content distribution packets flowchart

5.4.4.2 Least Number of Hops First (LNHF)

The LNHF was implemented creating an auxiliary architecture responsible for identifying the next packet to be forwarded and to keep the storage content and packets information such as the number of transmissions and number of hops updated.

Auxiliary Structures

Figure 5.22 gives an overview of the structures used to implement this architecture. The first one is a hash table which has as a key element, the packet's hash, and as a data element a pointer to the internal memory where information about that packet is stored. The second one is a double-linked list of pointers to internal memory where objects of type `Node` are stored with informations about stored packets. These structures are used mostly for two reasons. The first one is to provide an optimized way to identify the next packet to send according to a specific parameter, which is achieved through the use of a linked list always sorted according to this parameter. The second one is to quickly access an internal `Node` associated to a certain packet's hash and collect or change internal information about it.

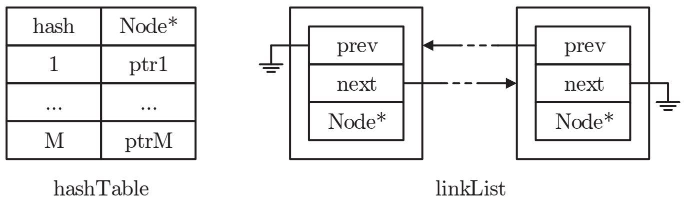


Figure 5.22: Helix integration HandlerLNHF implemented structures

As previously mentioned, multiple objects are created and stored in the internal memory, and each one contains several information about a specific content distribution packet already in storage identified by a unique hash. Each time that a node receives a packet that it does not have yet, it creates an object of type `Node` associated with the received packet and updates the handling structures. If a node already exists, it only updates the object and the structures if necessary. Thus, the number of entries of `hashTable` and `linkLinkList` must be equal to the number of content distribution packets stored by this node.

Implementation Scheme

Figure 5.23 shows an example of the implemented scheme and its behavior to select the packet to be forwarded. The scheme varies according to the node type. If it is an OBU, the double-linked list is sorted according to the number of hops (`nHops`), the element pointed by `head` the one with less number of hops. When multiple elements have an equal number of hops, the number of transmissions is the tie-breaking criteria. On the other hand, if it is an RSU the list is sorted according to the number of transmissions (`nTx`), the first element being the one with less transmissions.

Regarding this previous analysis, it is possible to explain the example of Figure 5.23. If the sender node is an OBU, the first packet to be peeked from storage will be the packet with `hash` number `4`, since it has the lowest number of hops among all the stored packets ($2 < 4 < 8$). Thus, the first element of the `linkLinkList` is the one associated with the `Node` pointed by `ptr4`. The next element of the `linkLinkList` is the `Node` associated with the packet which has the second lower number of hops that is the packet pointed by `ptr1`. In the third position of peeking is the packet which has the same number of hops than the last element, but with a lower number of transmissions, which is the packet pointed by `ptr9`. Finally, the last element is the packet which has the highest number of hops and the highest number of transmissions among the packets with an equal number of hops. This packet is associated with the `Node` pointed by `ptr6`.

On the other hand, if the sender node is an RSU, the first packet to be peeked from storage will be the packet with `hash` number `9`, since it has the lowest number of transmissions among all the stored packets ($1 < 4 < 5 < 9$). Thus, the first element of the `linkLinkList` is the one associated with the `Node` pointed by `ptr9`. The next element of the `linkLinkList` is the `Node` associated with the packet which has the second lowest number of transmissions that is the packet pointed by `ptr1`. In the third position of peeking is the packet which has the third lowest number of transmissions, which is the packet pointed by `ptr6`. Finally, the last element is the packet with has the highest number of transmissions among the packets. This packet is associated with the `Node` pointed by `ptr4`.

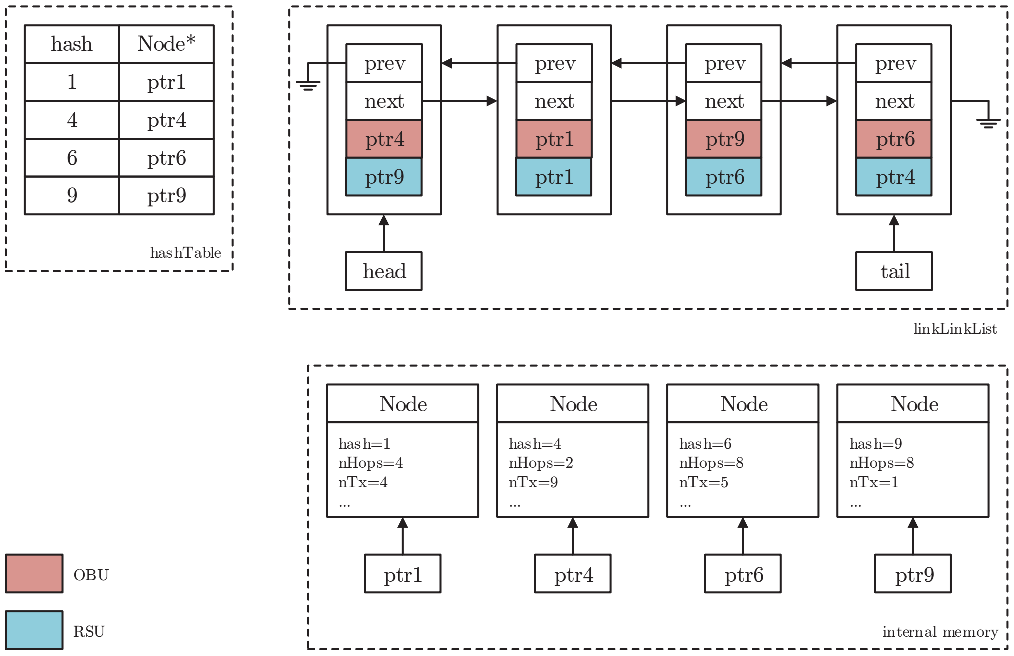


Figure 5.23: Helix integration HandlerLNHF implementation scheme

Implemented Code

In order to implement the LNHF strategy, a new class of objects was created which is illustrated by Figure 5.24. As mentioned before, the Routing module creates an object (`handlerLNHF`) of this class to perform the routing decisions according to this strategy. Internally, the `HandlerLNHF` class creates a nested class called `Node`. Each object of this nested class must be associated with a specific content distribution packet identified by a unique hash.

Regarding the LNHF strategy, the most important parameters of the `Node` class are the packet identifier (`hash`), number of transmissions (`nTx`) and hops (`nHops`) associated with this packet. To access and manipulate these members a set of thread-safe methods were developed. The `HandlerLNHF` class is responsible for the implementation of the support structures and also for their communication with the internal memory where `Node` objects are stored. This class also has access to the Storage module to have access to some specific information related to a packet. The hash table is implemented using the well-known *map* template of C++, and it is called `hashTable`. Furthermore, a double-linked list is implemented using another template of C++, *list*, and it is called `linkLinkList`.

To manipulate and access these structures several thread-safe methods were developed, such as `add2LinkList` (to add an element to the `linkLinkList`), `add2HashTable` (to add an element to the `hashTable`), and `sortLinkList` (to sort the `linkLinkList`). The last one is performed according to the node's type and uses one of two auxiliary comparison methods: `linkListCompOBU`, if it is an OBU or, if it is an RSU, `linkListCompRSU`. These methods are used to compared the number of hops/transmissions between adjacent nodes during the sorting procedure. Since `Node` is a nested class of `HandlerLNHF` the methods to access and manipulate from outside are implemented in outer class. The most important methods are the `insertNode` (to insert a new `Node`), `removeNode` (to remove a `Node`), `updateNode` (to update an existent `Node` according to new information), `incnTxNode` (to increment the number of transmissions of the packet associated with this `Node`), and `getNode` (to collect information about a `Node`).

It is possible to manipulate the hash table and the linked list at the same time since they have independent mutex, but a `Node` object can only be manipulated by one entity at the same time since it has a unique internal mutex.

Once the overall implementation is discussed, the two main procedure of content distribution schemes and how they were implemented will be described.

Data Packet Forwarding Decision

Figure 5.25 describes the forwarding decision procedure of data packets. This is basically the description of the `runRounting` thread of the Routing module and its behavior is equally independent from the node's type. The procedure is executed with a periodicity of `CYCLE_-DELAY` microseconds (default value is 200 ms), and the end of the procedure is immediately before the start of this sleep. Thus, if a node has neighbors which are all OBUs, and has packets in storage, a hash from `linkLinkList` is peeked. This operation is performed invoking the `getFirstAvailableNode` method which returns the hash associated with the first element of `linkLinkList` which is assumed to be updated accordingly with a specific parameter (number of hops, if node is an OBU, number of transmissions, if it is an RSU). If a hash was retrieved, there is an attempt to collect the packet associated with that hash from storage through `peek_hash` method. If there is no packet associated with this hash in storage, the information associated with it is removed from structures `linkLinkList` and `hashTable`, and also from the internal memory.

After that the packet is peeked from storage and, if it is not in the OnHold table, the service ID associated with it is analyzed. Otherwise, the procedure jumps to its end. If it is not a content distribution packet, all elements associated with this hash are removed from the internal structures and the procedure ends. Otherwise, the packet is updated in the following fields, `prev_EID` with the ID of the sender node and, if the number of hops registered in packet's header is lower than the one recorded in the internal memory (`Node`'s object), the `numNeigh` field (which represents the packet's number of hops) is updated with the recorded value. After that, the packet is sent in broadcast to the node's vicinity, and the

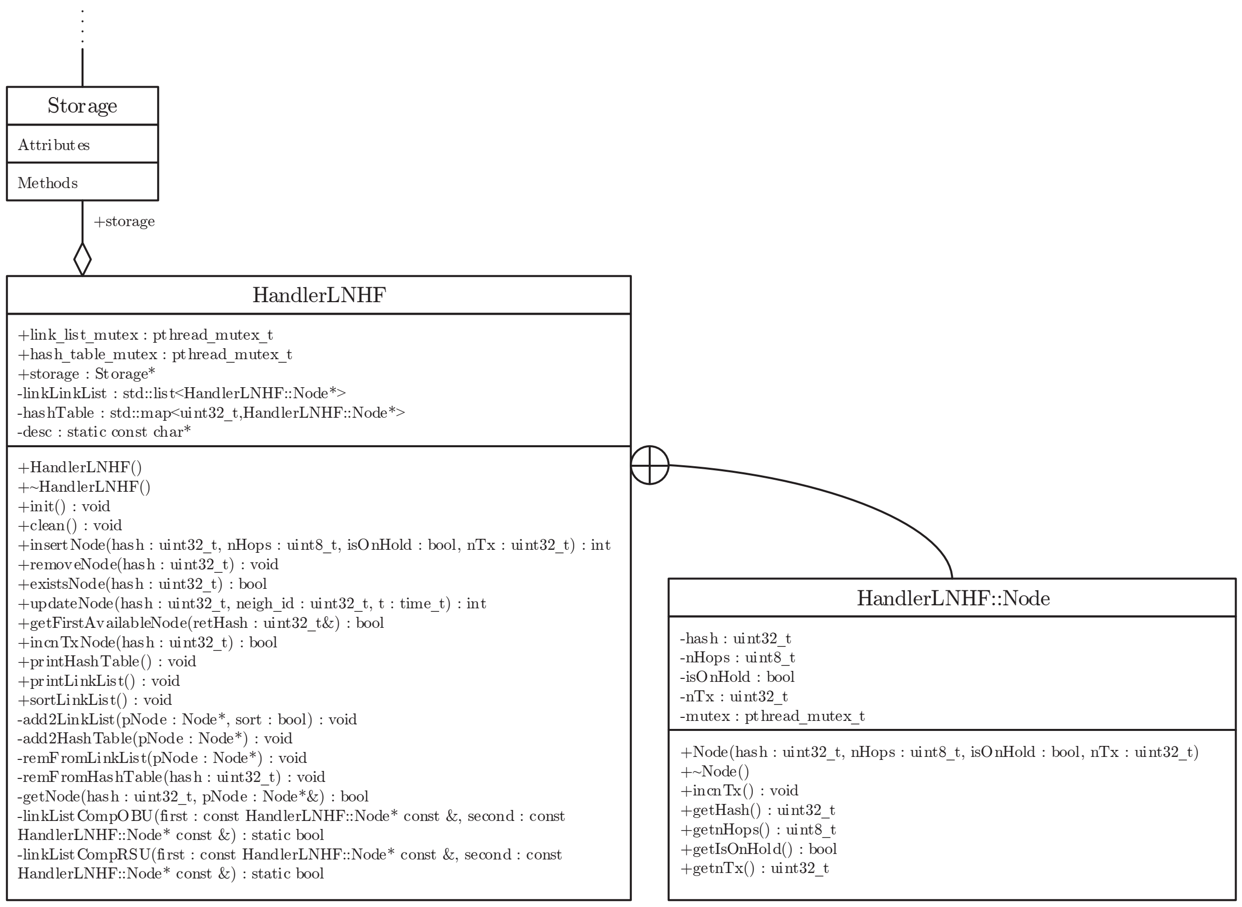

Storage
Attributes
Methods
+storage
HandlerLNHF
+link_list_mutex : pthread_mutex_t
+hash_table_mutex : pthread_mutex_t
+storage : Storage*
-linkLinkList : std::list<HandlerLNHF::Node*>
-hashTable : std::map<uint32_t,HandlerLNHF::Node*>
-desc : static const char*
+HandlerLNHF()
+~HandlerLNHF()
+init() : void
+clean() : void
+insertNode(hash : uint32_t, nHops : uint8_t, isOnHold : bool, nTx : uint32_t) : int
+removeNode(hash : uint32_t) : void
+existsNode(hash : uint32_t) : bool
+updateNode(hash : uint32_t, neigh_id : uint32_t, t : time_t) : int
+getFirstAvailableNode(retHash : uint32_t&) : bool
+incnTxNode(hash : uint32_t) : bool
+printHashTable() : void
+printLinkList() : void
+sortLinkList() : void
-add2LinkList(pNode : Node*, sort : bool) : void
-add2HashTable(pNode : Node*) : void
-remFromLinkList(pNode : Node*) : void
-remFromHashTable(hash : uint32_t) : void
-getNode(hash : uint32_t, pNode : Node*&) : bool
-linkListCompOBU(first : const HandlerLNHF::Node* const &, second : const HandlerLNHF::Node* const &) : static bool
-linkListCompRSU(first : const HandlerLNHF::Node* const &, second : const HandlerLNHF::Node* const &) : static bool
HandlerLNHF::Node
-hash : uint32_t
-nHops : uint8_t
-isOnHold : bool
-nTx : uint32_t
-mutex : pthread_mutex_t
+Node(hash : uint32_t, nHops : uint8_t, isOnHold : bool, nTx : uint32_t)
+~Node()
+incnTx() : void
+getHash() : uint32_t
+getnHops() : uint8_t
+getIsOnHold() : bool
+getnTx() : uint32_t

Figure 5.24: Helix integration HandlerLNHF sub-module collaboration diagram

nTx filed of internal memory's object is incremented, and linkLinkList is sorted since one of the comparison parameters was updated. Before finishing, the logging variables such as packets_transmitted_total and packets_transmitted_per_timestamp are incremented.

Data Packet Reception

In Figure 5.26 the procedure carried out when a data packet is received from the RX module is illustrated. Since the procedure varies according to the node type, the analysis is divided in two. If the receiver is an OBU and the packet was received from a WAVE interface, the node ID is added to the packet's hop list. After that, the existence of a Node in internal memory associated with the packet received is tested. If there is any Node, and the packet was received from a valid neighbor, a new one is created using the insertNode method. This method creates a new object of type Node and fills its members with the packet's identifier (hash) and the number of hops (numNeigh) and transmissions (nTx) set to 0, and also adds new entries to the hash table and to the double-linked list which both has a pointer to the internal memory address where the created Node is stored. After that, the linkLinkList is sorted since a new element was introduced. Otherwise, if a Node associated with the received packet already exists, and the packet number of hops field numNeigh is greater than the value

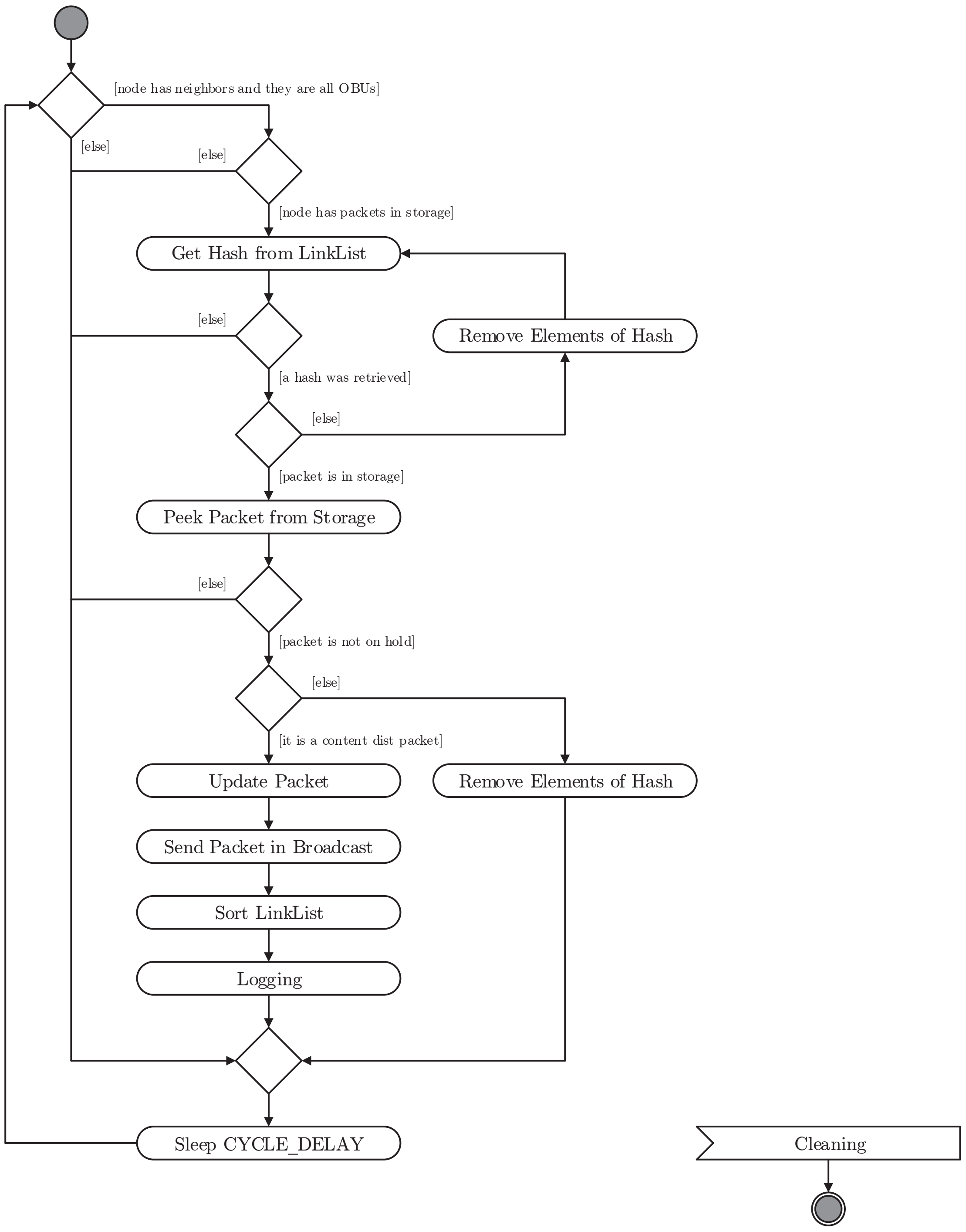


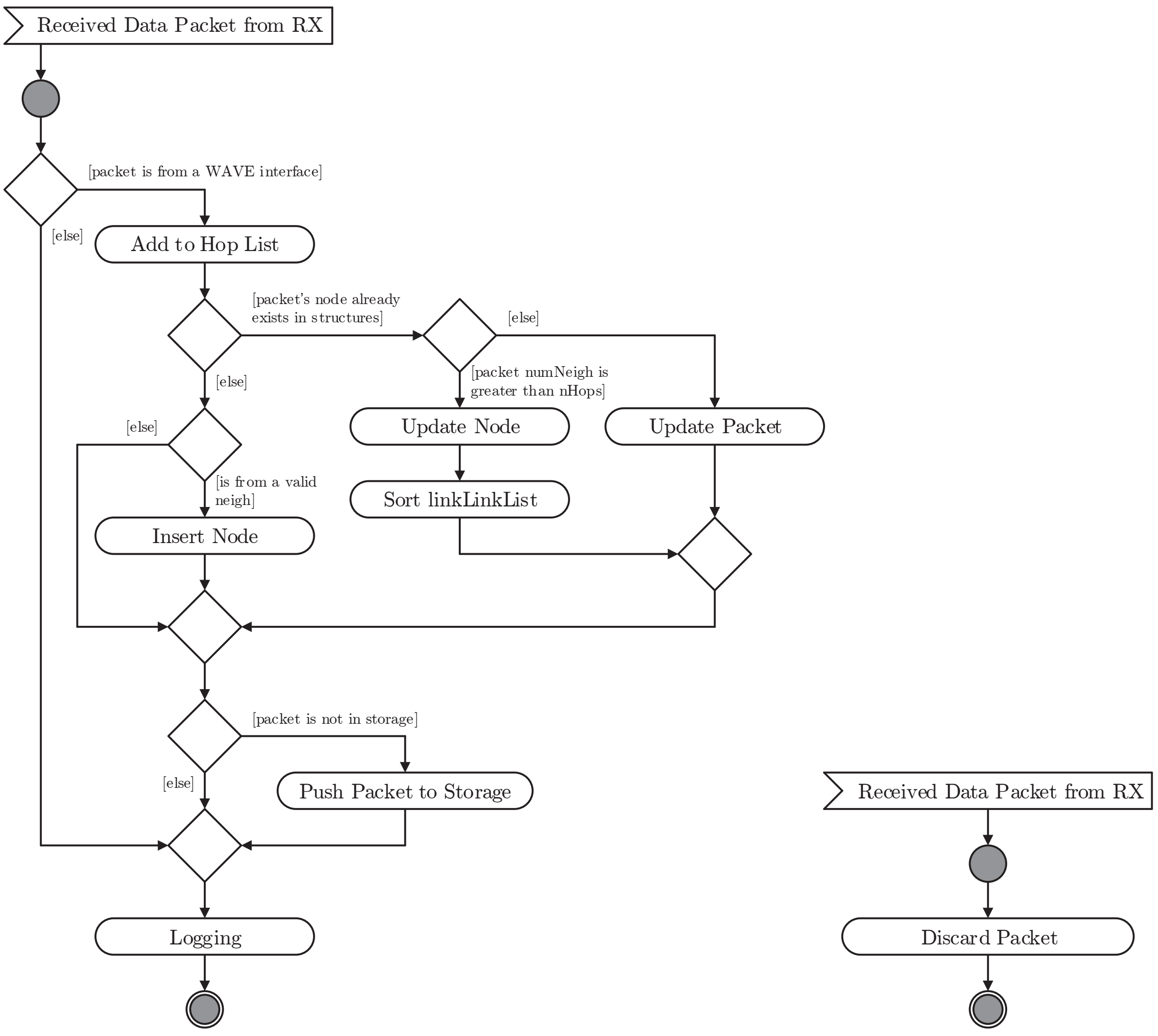
Figure 5.25: Helix integration LNHF strategy data packet forwarding decision flowchart

of the number of hops recorded (`nHops`), the `numNeigh` attribute of `Node` is updated with the received value. If not, the packet's field is updated with the recorded value. If `nHops` was updated, the `linkLinkList` is sorted.

When the internal structures are updated or a new `Node` and respective entries of structures are inserted, and if the packet is not in storage, it is stored. Otherwise, the procedure jumps to the logging phase. Thus, when the node does not have store the received packet the counters of the number of packets stored and packets good receives are incremented (`packets_stored_total`, `packets_stored_per_timestamp`, and `packets_recv_-good_per_timestamp`). However, if an already stored packet was heard, the number of wrong received packets is incremented (`packets_recv_bad_per_timestamp`). In both cases, the number of listened packets in this timestamp is also incremented (`packets_listened_per_-timestamp`). On the other hand, if the receiver node is an RSU, the packet is discarded since the dissemination procedure is always downstream (from RSUs to OBUs or among OBUs).

Initial Generation of Content Distribution Packets

The procedure to initially store packets in RSUs is illustrated in Figure 5.27. Once the number of packets is defined, all packets are created and stored. First the packet's header is initialized and a packet is created (join of header and payload). After that, the packet is pushed to storage and a `Node` in internal structures is created. Since a new packet was pushed, `linkLinkList` is updated.



Received Data Packet from RX
[packet is from a WAVE interface]
[else]
Add to Hop List
[packet's node already exists in structures]
[else]
[else]
[packet numNeigh is greater than nHops]
Update Node
Update Packet
[else]
[is from a valid neigh]
Sort linkLinkList
Insert Node
[packet is not in storage]
[else]
Push Packet to Storage
Logging
Received Data Packet from RX
Discard Packet

(a)

(b)

Figure 5.26: Helix integration LNHF strategy data packet reception in (a) an OBU and (b) an RSU flowcharts

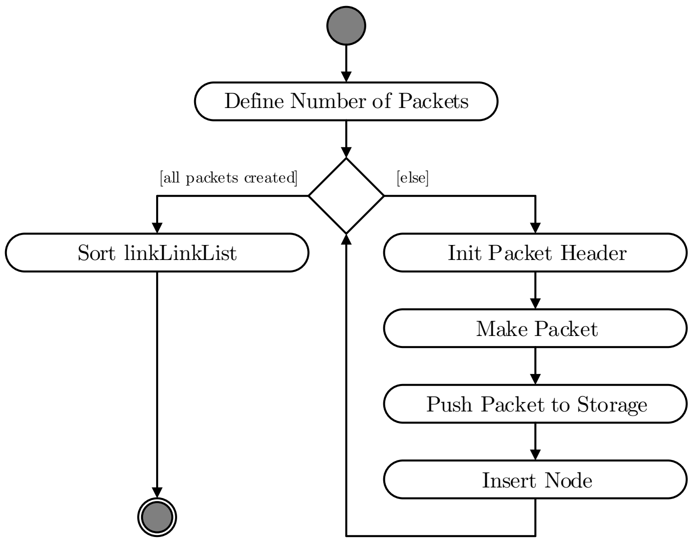


Figure 5.27: Helix integration LNHF strategy generation of content distribution packets flowchart

5.4.4.3 Local Rarest Bundle First (LRBF)

The LRBF was implemented by creating an auxiliary architecture responsible for identifying the next packet to be forwarded implementing all the features and functionalities to achieve this goal. As its major feature, the LRBF implementation added new control packets responsible for advertising the storage content to the node's neighbors in order to perform the forwarding decision. Moreover, this implementation introduces the possibility of spreading multiple files simultaneously, although this last feature is not the core of this implementation but only an add-on.

Auxiliary Structures

Figure 5.28 describes an overview of the structures used in the implementation of this strategy. The first one is a hash table which has the same elements as the previously presented in Figure 5.22, whereby its key element is the packet's hash, and the data element is a pointer to the internal memory where information about that packet is stored. Likewise the previous one, the second used structure is also similar to the one used in the LNHF strategy, being a double-link list of pointers to internal memory where objects of type `Node` with informations about stored packets are stored. The number of entries of `hashTable` and `mainLinkList` must be equal to the number of content distribution packets stored by a node.

Compared to the LNHF strategy a new structure was added to map the files that are being spread through the network. It is also an associative container, but now it stores elements formed by a combination of a key value, `fileID`, and a mapped value, a pair of two values: number of packets stored of that file (`nPacketsStored`), and total packets of that file (`totalPackets`). Since each node must know what are the most common bundles in its neighbors, a new structure was added to `Node` class, the `listPairs`. This is a double-link list of a pair of values: ID of the neighbor (`neighID`), and the time when the information indicating that the neighbor identified by `neighID` had the packet associated with this `Node` (`time_last`) was collected.

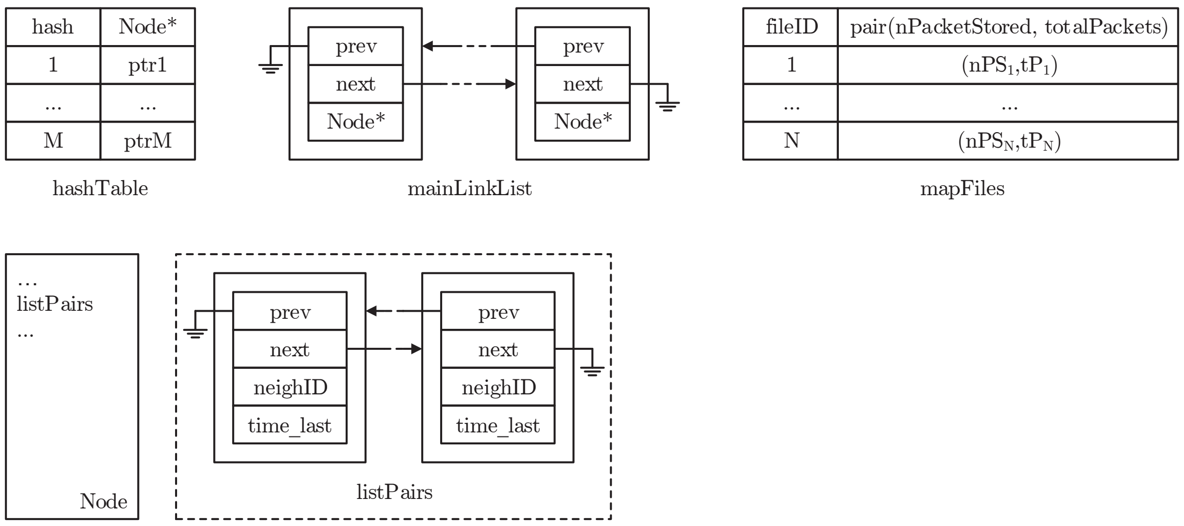


Figure 5.28: Helix integration HandlerLRBF implemented structures

Similarly to the LNHF strategy, these structures are used mostly for two reasons. The first one is to provide an optimized way to identify the next packet to send, and the second one is to have fast access to an internal `Node` associated to a specific packet's hash. The new table maps the files which are being spread, whereby it allows to detect when a node download all the packets of a certain file. Since each `Node` has a content distribution packet associated, the `listPairs` (which is the sorted list of `Nodes` according to the strategy criteria) allows to identify which neighbors have a specific packet. The timestamp of when the control data was collected (`time_last`) is used in periodical refreshes to the `listPairs` since, if a neighbors moves away from the node's vicinity, the information should be discarded in order to perform a more accurate forwarding decision.

Implementation Scheme

An example of the implemented scheme and its behavior to select the packet to be forwarded is shown in Figure 5.29. The double-link list (`mainLinkList`), responsible to indicate the packet to be sent, is sorted according to the number of neighbors which have a specific packet (size of the `listPairs` associated with this packet). In the first position of the (`mainLinkList`), which is pointed by `head`, it is stored a pointer to the `Node` which has the shortest `listPairs` list, meaning that more neighbors are in need of that packet. When multiple elements of the `mainLinkList` have associated an equal size of `listPairs`, the number of transmissions is the tie-breaking criteria. As mentioned before, `listPairs` of each node store information about which neighbors have the packet associated with this `Node`, whereby the one with smallest size means that it is the most lacking packet in the node's vicinity.

Regarding this previous analysis, it is possible to explain the example of Figure 5.29. In the internal memory there is information about four packets (already stored by the sender node) which is organized in objects of type `Node`. The `listPairs` list stores information collected from neighboring nodes which allows the understanding of how many neighbors already have a specific packet (each element of this list is associated with a neighbor). Thus, the size of the `listPairs` is directly associated with the most lacking packet within the sender node vicinity. The `mainLinkList` is a double-link list which allows a faster selection of the next packet to be forwarded, whereby it is sorted according to the size of `listPairs` list. Since the `Node` associated with the packet with `hash` number `4` is the most lacking packet (shortest size of `listPairs`), a pointer to it is stored in the first position of `mainLinkList` list. Following the same reasoning, the next pointed `Node` is the one identified by `hash 1` and pointed by `ptr1`. Since the number of transmission is the tiebreaker criteria, the next packet to be sent is the one pointed by `ptr9` and the last one pointed by `ptr6`.

Implemented Code

In order to implement the LRBF strategy a new class of objects was created. This is described in Figure 5.30. The Routing module creates an object (`handlerLRBF`) of this class to perform the routing decisions according to this strategy. Similar to `HandlerLNHF`, the `HandlerLRBF` class creates a inner class called `Node`. Each object of `Node` class has associated a specific content distribution packet identified by an unique hash.

To perform the forwarding decisions according to the LRBF strategy, the `Node` class has the following members: `hash`, `fileID`, `listPairs`, and `nTx`. A set of thread-safe methods were developed to access and manipulate these members. Among them, the most important are `incnTx` (to increment number of transmissions method of the class `Node`), `updateListPairs`, `refreshListPairs`, `getListPairsSize`, and `listPairsComp`. The `updateListPairs` allows

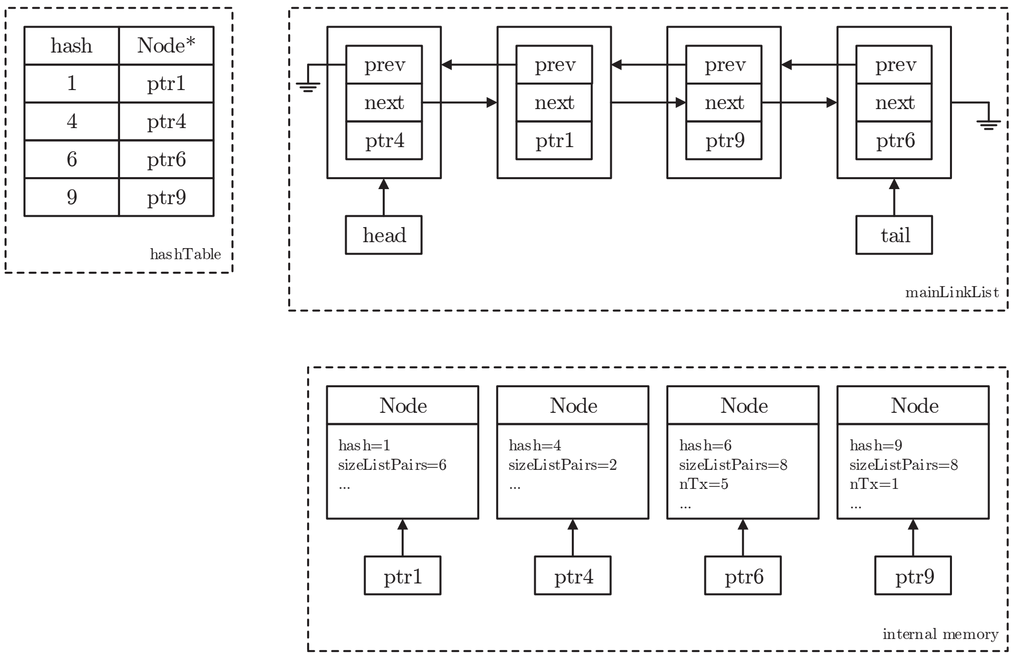


Figure 5.29: Helix integration HandlerLRBF implementation scheme

the update of the `listPairs` list with a new element, or updating the `time_last` parameter if an element of `neigh_id` already exists. On the other hand, `refreshListPairs` is periodically called to clean dated elements of `listPairs`. An element is considered dated if it was not updated for a period longer than `ELEMENT_VALID_TIME` seconds (default value is 60 s). `getListPairsSize` is fundamentally used by the sort function of `mainLinkList`, since it relies mostly on the size of `listPairs` of each node within it. Finally, `listPairsComp` is used as the comparison function of the sort method of `listPairs` which sets the older updated element at the head of the list. This approach improves the refreshing procedure that will be further discussed in this document. Furthermore, two class constructors were implemented, which allow the creation of new objects in one of two ways: (i) an object with an empty `listPairs`, or (ii) with one element in `listPairs`.

To implement the support structures, a new class called `HandlerLRBF` was developed which is responsible for the communication between them and objects of inner class `Node` stored in the internal memory. This class also has access to the Storage module in order to have access to specific information related to a packet. The well-known template of C++ *map* is used to implement the `hashTable` and `mapFiles` structures, the `mainLinkList` is implemented using the *list* template.

Among the several thread-safe methods developed to access and manipulate the internal structures, the most relevant are `peekMainListHashes`, `add2MainList`, `add2HashTable`, `add2MapFiles`, `runRefresh`, and `sortLinkList`. Since `Node` is an inner class of `HandlerLRBF`, the methods to access and manipulate from the outside are implemented in the outer class. The most important methods are the `insertNode`, `insertEmptyNode`, `getFirstAvailableNode`, `removeNode`, `updateNode`, `incnTxNode`, and `getNode`. Here is not given a complete description of these methods since it will be done in the following topics as justified.

It is possible to manipulate the `hashTable`, `mapFiles`, and `mainLinkList` at the same time since they have an independent mutex, but a `Node` object can only be manipulated by one entity at the same time since it has an unique internal mutex.

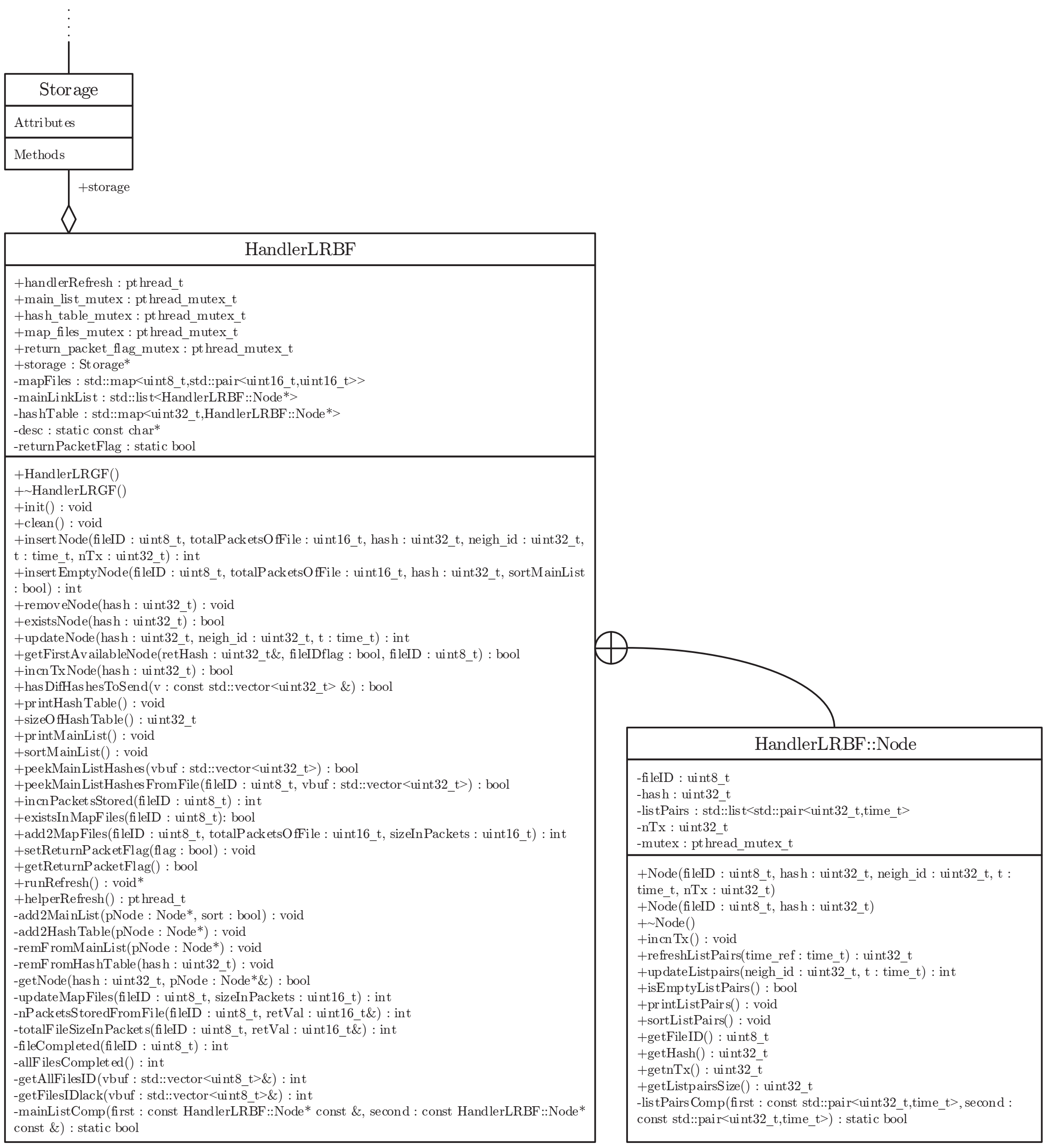

Storage
Attributes
Methods
+storage
HandlerLRBF
+handlerRefresh : pthread_t
+main_list_mutex : pthread_mutex_t
+hash_table_mutex : pthread_mutex_t
+map_files_mutex : pthread_mutex_t
+return_packet_flag_mutex : pthread_mutex_t
+storage : Storage*
-mapFiles : std::map<uint8_t,std::pair<uint16_t,uint16_t>>
-mainLinkList : std::list<HandlerLRBF::Node*>
-hashTable : std::map<uint32_t,HandlerLRBF::Node*>
-desc : static const char*
-returnPacketFlag : static bool
+HandlerLRGF()
+~HandlerLRGF()
+init() : void
+clean() : void
+insertNode(fileID : uint8_t, totalPacketsOfFile : uint16_t, hash : uint32_t, neigh_id : uint32_t, t : time_t, nTx : uint32_t) : int
+insertEmptyNode(fileID : uint8_t, totalPacketsOfFile : uint16_t, hash : uint32_t, sortMainList : bool) : int
+removeNode(hash : uint32_t) : void
+existsNode(hash : uint32_t) : bool
+updateNode(hash : uint32_t, neigh_id : uint32_t, t : time_t) : int
+getFirstAvailableNode(retHash : uint32_t&, fileIDflag : bool, fileID : uint8_t) : bool
+incnTxNode(hash : uint32_t) : bool
+hasDifHashesToSend(v : const std::vector<uint32_t> &) : bool
+printHashTable() : void
+sizeOfHashTable() : uint32_t
+printMainList() : void
+sortMainList() : void
+peekMainListHashes(vbuf : std::vector<uint32_t>) : bool
+peekMainListHashesFromFile(fileID : uint8_t, vbuf : std::vector<uint32_t>) : bool
+incnPacketsStored(fileID : uint8_t) : int
+existsInMapFiles(fileID : uint8_t): bool
+add2MapFiles(fileID : uint8_t, totalPacketsOfFile : uint16_t, sizeInPackets : uint16_t) : int
+setReturnPacketFlag(flag : bool) : void
+getReturnPacketFlag() : bool
+runRefresh() : void*
+helperRefresh() : pthread_t
-add2MainList(pNode : Node*, sort : bool) : void
-add2HashTable(pNode : Node*) : void
-remFromMainList(pNode : Node*) : void
-remFromHashTable(hash : uint32_t) : void
-getNode(hash : uint32_t, pNode : Node*&) : bool
-updateMapFiles(fileID : uint8_t, sizeInPackets : uint16_t) : int
-nPacketsStoredFromFile(fileID : uint8_t, retVal : uint16_t&) : int
-totalFileSizeInPackets(fileID : uint8_t, retVal : uint16_t&) : int
-fileCompleted(fileID : uint8_t) : int
-allFilesCompleted() : int
-getAllFilesID(vbuf : std::vector<uint8_t>&) : int
-getFilesIDlack(vbuf : std::vector<uint8_t>&) : int
-mainListComp(first : const HandlerLRBF::Node* const &, second : const HandlerLRBF::Node* const &) : static bool
HandlerLRBF::Node
-fileID : uint8_t
-hash : uint32_t
-listPairs : std::list<std::pair<uint32_t,time_t>
-nTx : uint32_t
-mutex : pthread_mutex_t
+Node(fileID : uint8_t, hash : uint32_t, neigh_id : uint32_t, t : time_t, nTx : uint32_t)
+Node(fileID : uint8_t, hash : uint32_t)
+~Node()
+incnTx() : void
+refreshListPairs(time_ref : time_t) : uint32_t
+updateListpairs(neigh_id : uint32_t, t : time_t) : int
+isEmptyListPairs() : bool
+printListPairs() : void
+sortListPairs() : void
+getFileID() : uint8_t
+getHash() : uint32_t
+getnTx() : uint32_t
+getListpairsSize() : uint32_t
-listPairsComp(first : const std::pair<uint32_t,time_t>, second : const std::pair<uint32_t,time_t>) : static bool

Figure 5.30: Helix integration HandlerLRBF sub-module collaboration diagram

Start and End of Dissemination

There are two key flags that define when a node can send data and/or control packets. The one responsible for allowing the broadcast of advertisement packets is called `is2sendAdv` and is initialized as false in OBUs and true in RSUs. If this flag is enabled, a node can broadcast advertisement packets to its neighbors; otherwise it must until the `is2sendAdv` change its value to true. The other flag, `returnPacketFlag`, is initialized as false in both OBUs and RSUs and, when enabled, allows the broadcast of content distribution data packets.

Thus, when Helix starts running, only RSUs send advertisement packets and any node sends data packets. Figure 5.31-(a) illustrates the procedure to enable the broadcast of data packets. The node which is able to send advertisement packets(`is2sendAdv` flag is active), periodically sends advertisement packets announcing its storage content. At the beginning this node can only be an RSUs but after a time, can also be an OBU, since at the beginning of the dissemination only the RSUs known the existence of a content to be spread trough the network. When an OBU which does not have any content yet receives this advertisement, it gets to know that there is content to be downloaded, whereby it updates the internal structures and can start the broadcast of advertisement packets (changes its `is2sendAdv` flag to true). So, it advertises that it does not have any packet of such content. When the first node receives this announcement, it understands that there is at least one neighbor which does not have any data associated with the content under dissemination. Moreover, if the advertisement reports incomplete content and the receiver node has data useful to the advertiser (data that it does not have), it changes its `returnPacketFlag` to true to be able to send useful data packets in broadcast.

On the other hand, as described in Figure 5.31-(b), a node should end the process of dissemination when it does not have any neighbors or if the current neighbors have completed the download of the content. Thus, when these conditions are reached, the refresh thread empties `listPairs` and sets the `returnPacketFlag` to false, unauthorizing the sending of content distribution packets.

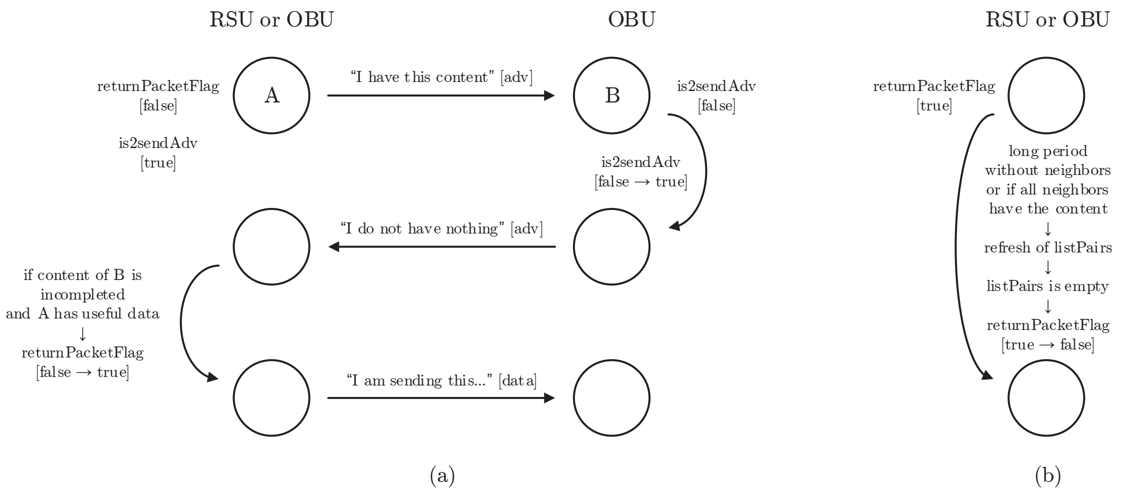

RSU or OBU
OBU
RSU or OBU
returnPacketFlag [false]
A
is2sendAdv [true]
"I have this content" [adv]
B
is2sendAdv [false]
is2sendAdv [false → true]
"I do not have nothing" [adv]
if content of B is incompleted and A has useful data ↓ returnPacketFlag [false → true]
"I am sending this..." [data]
(a)
returnPacketFlag [true]
long period without neighbors or if all neighbors have the content ↓ refresh of listPairs ↓ listPairs is empty ↓ returnPacketFlag [true → false]
(b)

Figure 5.31: Helix integration HandlerLRBF (a) start and (b) end of dissemination

Refresh Thread

As mentioned before, the elements in `listPairs` have a limited lifetime of `ELEMENT_-`

VALID_TIME seconds. Thus, a thread was created in order to, in a periodical way, check the validity of listPairs's elements and erase the expired ones.

Figure 5.32 describes how this thread is implemented. At the beginning of the procedure, the actual time (time_ref) is collected which is the time reference to evaluate the validity of each element. For each element of the mainLinkList, the refresh of listPairs using the refreshListPairs method is done. This method runs through all elements of listPairs and, if the information is no longer valid (time_ref-time_last is lower than ELEMENT_-VALID_TIME), this element will be erased from listPairs. Since the listPairs is sorted according to time_last value, the iterative process continues until it reaches a valid element because all the further elements are valid.

After that, if any element from listPairs was removed, the list must be sorted whereby flag is2sort is set to true and, if listPairs is empty, flag isAllEmpty is enabled to report the list emptiness. If one or both conditions are not fulfilled, the respective flags are not enabled. After that, if listPairs is empty (isAllEmpty is enabled) the broadcast of packets is interrupted (returnPacketFlag is disabled). This last procedure guarantees the end of the dissemination process when all the node's neighbors have the content (see Figure 5.31). If necessary, the mainLinkList is sorted since at least one element of listPairs of a certain Node was removed.

This thread is executed with a random periodicity between MIN_REFRESH_PERIOD and MAX_REFRESH_PERIOD seconds to avoid possible starvation caused by fixed periodicity. However, these two macros should be kept closer in order to set a more accurate refreshment period.

Data Packet Forwarding Decision

The flowchart presented in Figure 5.33 illustrates how the forwarding decision of data packets in this strategy is done. The thread runRouting is running this procedure to implement the LRBF strategy, and its behavior is equally independent from the node's type. This procedure has an approximate periodicity of CYCLE_DELAY microseconds (default value is 200 ms).

So, if the sender node has neighbors and all are OBUs, and it has packets in storage, the method getFirstAvailableNode is invoked to peek the hash that will be loaded from storage. This method returns the hash associated with the first element of the sorted mainLinkList (remember that this list is sorted according to the size of each Node's listPairs). A hash is only retrieved if the sender node has permission to do it (canReturnPacketFlag is active). Furthermore, if there is an element of mainLinkList associated to a packet that is not stored, that Node is added to the delete list to be deleted at the end, and the procedure iterates to the next Node to collect another hash to be retrieved.

After that, if a hash was retrieved and it identifies a content distribution packet, the packet is updated (fields prev_EID and numNeigh - similarly to LNHF strategy), being then broadcasted. If the picked hash does not identify a content distribution packet, the Node associated with it is deleted from internal structures and memory using the removeNode method. Once the packet is sent, the number of transmissions of Node is incremented and the mainLinkList is sorted. At the end of the procedure, the logging variables packets_-transmitted_total and packets_transmitted_per_timestamp are incremented.

The runRouting thread runs until a cleaning signal is sent.

Data Packet Reception

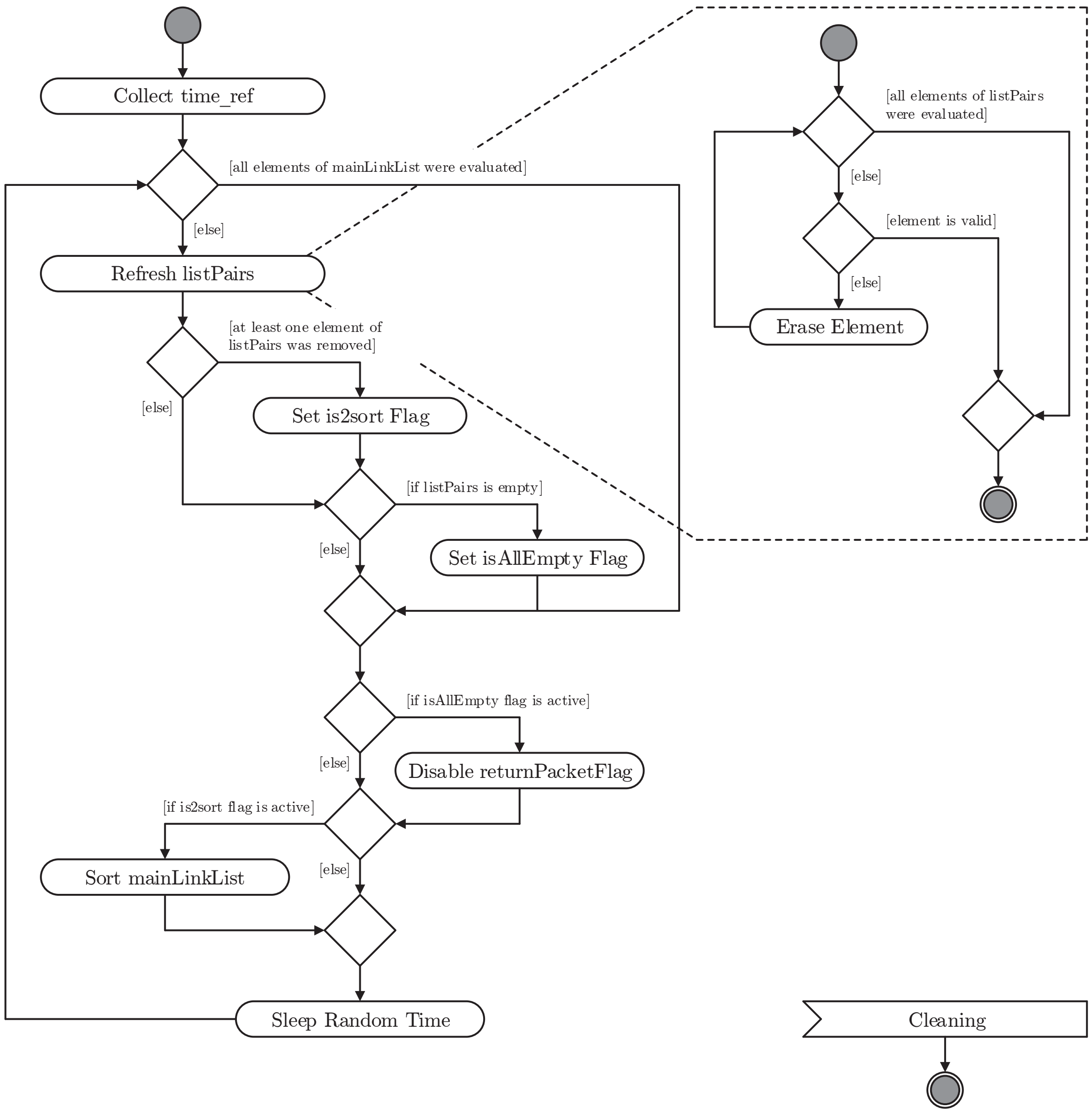


Figure 5.32: Helix integration LRBF strategy refresh thread flowchart

The procedure carried out when a data packet is received from the RX module is illustrated in Figure 5.34. If the receiver node is an RSU, the packet is discarded since the dissemination procedure is always downstream (from RSUs to OBUs or among OBUs). On the other hand, if the receiver is an OBU and the packet was received from a WAVE interface, the packet's hop list is updated with the ID of the receiver node.

If there is any `Node` in internal structures associated with packets received, a new `Node` is inserted. There are two ways to insert this new `Node` according to the sender node's type. If it is received from an OBU, a completed `Node` is added using the method `insertNode`. This method creates a new `Node` and also initializes the `listPairs` with a new element composed

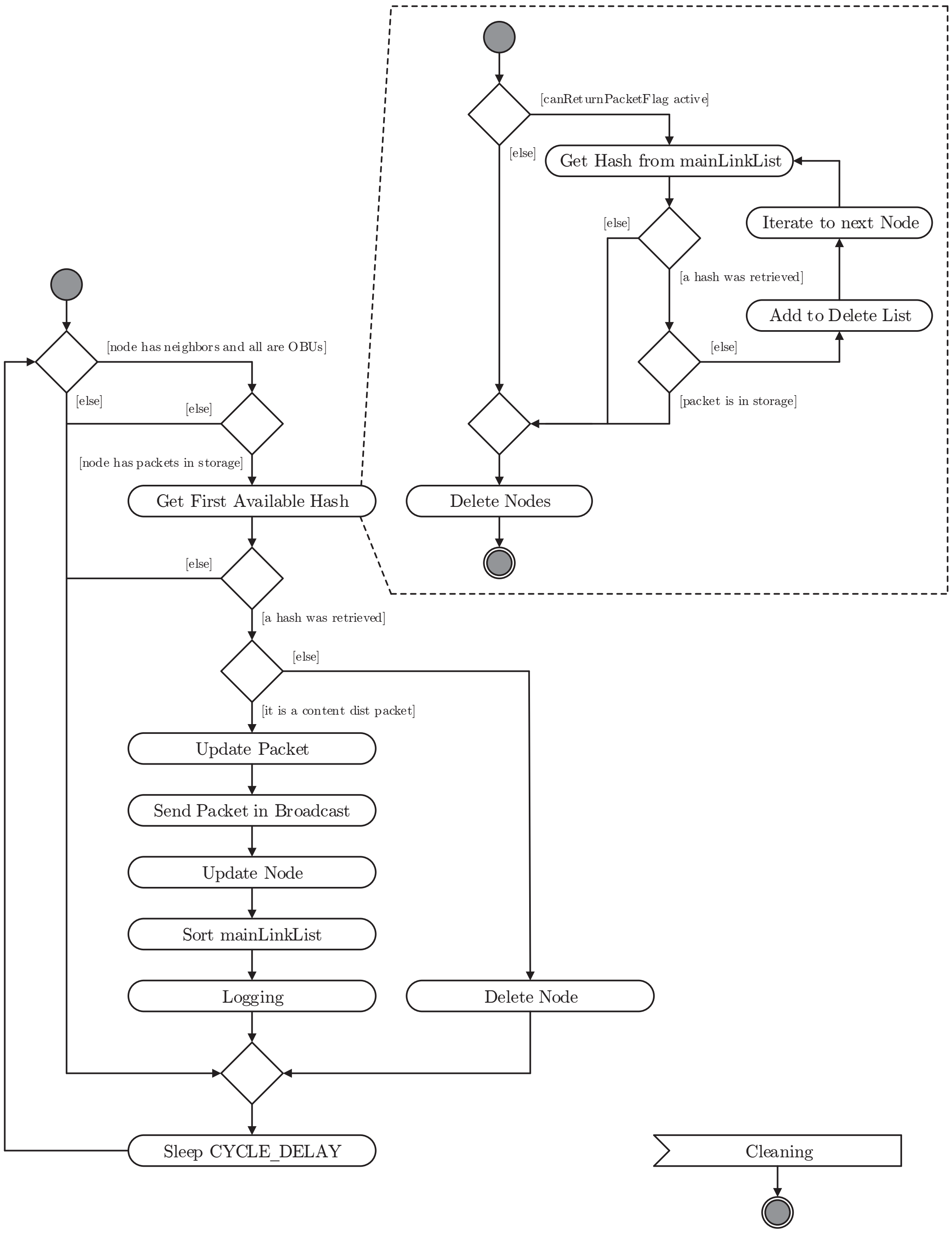


Figure 5.33: Helix integration LRBF strategy data packet forwarding decision flowchart

by the sender ID and actual timestamp (uses `Node`'s constructor with five arguments). On the other hand, if the sender node is an RSU, it is not important to use that information for the forwarding decision since a node only needs information about the content of its neighbors which are OBUs. Thus, a method to only create a new `Node` (and respectively create new entries in structures) with an empty `listPairs` was developed, and is called `insertEmptyNode`. This procedure allows the receiver node to add information about a new packet to structures without misrepresenting the forwarding decision with useless information. If the incoming packet added information about a new content to download, the receiver node must be able to start forwarding advertisement packets claiming for this content (`is2sendAdv` is enabled).

However, if it already exists a `Node` associated with the received packet in structures, and the sender node is a valid neighbor and an OBU, the `Node` is updated. To perform this updated the `updateNode` method is used. This method can do one of two things: (i) add a new element to `listPairs`, or (ii) update an existing element. If a new element is added, the size of `listPairs` changes, whereby the `mainLinkList` must be sorted; otherwise the procedure jumps to the intermediary stage where it joins the other branch of the flowchart.

Once the internal structures are updated or a new `Node` is inserted and respective entries of structures, if the packet is not in storage, it is stored and the number of stored packets of this file is updated in `mapFiles`. If it is, the procedure jumps to the logging phase. Then, if the node does not have the received packet, the total number of packets along with the amount of packets stored in this timestamp are incremented (`packets_stored_total` and `packets_-stored_per_timestamp`), and the number of good received packets (a node does not have it yet) is incremented (`packets_recv_good_per_timestamp`). Although, if a packet already stored was heard, the `packets_recv_bad_per_timestamp` is incremented. The number of heard packets in this timestamp (`packets_listened_per_timestamp`) is incremented in both cases.

Advertisement Procedures

Packet Structure

The LRBF strategy relies on the exchanging of advertisement messages announcing the content of the node's storage (see Figure 4.11). To implement these messages, advertisement packets were introduced and follow the structure illustrated by Figure 5.35. Such packet is an accumulated set of information about the content that is being spread through the network and the node has knowledge about it. Thus, in the LRBF content distribution scheme an advertisement packet starts with the total number of files known by the node followed by specific information about each file. This information has the file identifier, the total number of packets that compose this file and how many of them a node has already stored. Furthermore, if not all of the packets were collected by the node, the advertisements also have the list of hashes belonging to this file that are stored in the node's storage. This set of information is replicated for each known file. Every element of this packet has a size of 4 Bytes. In the following topics it is discussed how a node fills the advertisement packet and how the information is used by its receiver.

Forwarding Decision

The advertisement packet forwarding decision flowchart is presented in Figure 5.36. If `is2sendAdv` flag is active, advertisement packets will be sent, whereby the advertisement

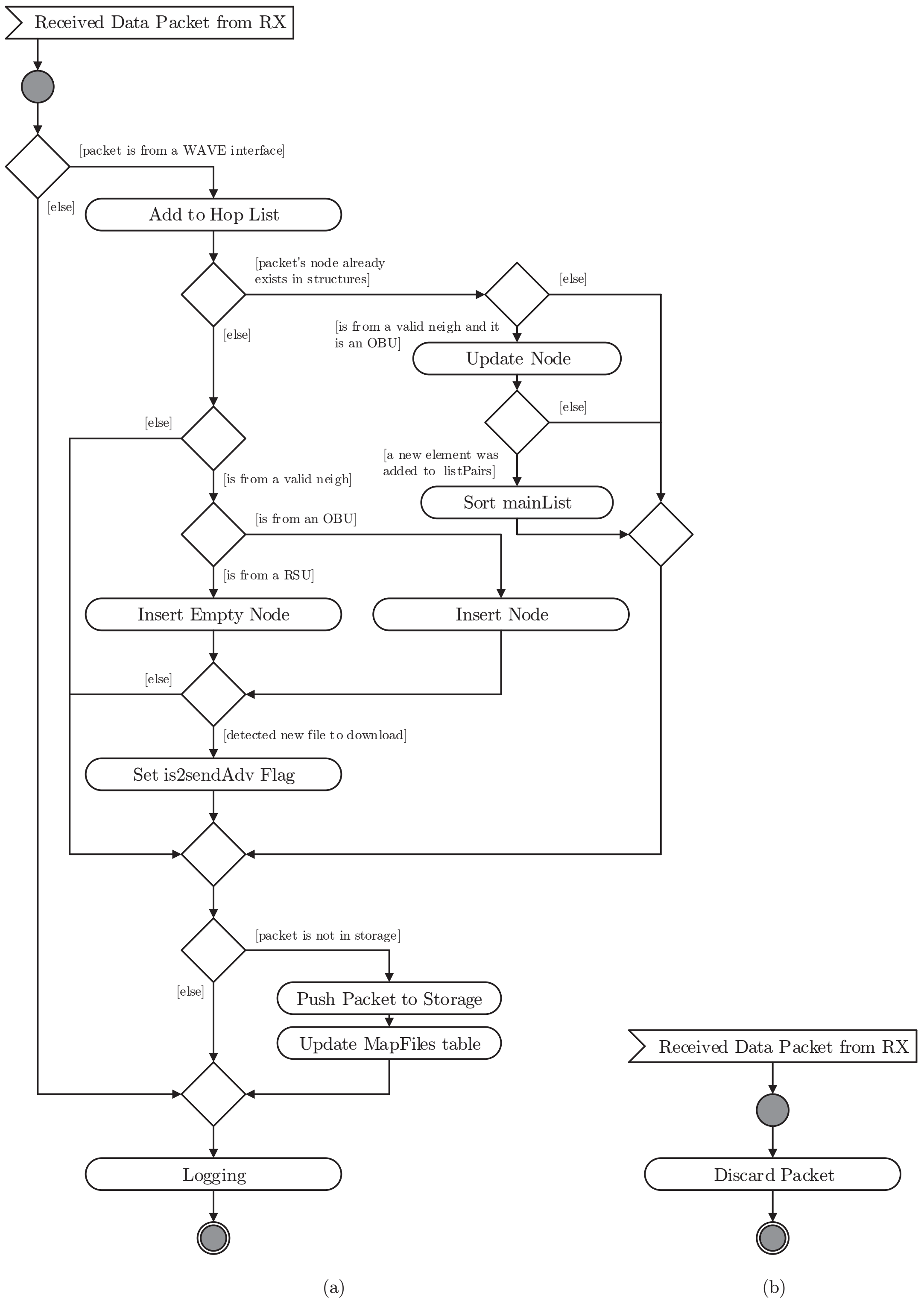

Received Data Packet from RX
[packet is from a WAVE interface]
[else]
Add to Hop List
[packet's node already exists in structures]
[else]
[else]
[is from a valid neigh and it is an OBU]
Update Node
[else]
[else]
[a new element was added to listPairs]
Sort mainList
[is from a valid neigh]
[is from an OBU]
[is from a RSU]
Insert Empty Node
Insert Node
[else]
[detected new file to download]
Set is2sendAdv Flag
[packet is not in storage]
[else]
Push Packet to Storage
Update MapFiles table
Logging
(a)
Received Data Packet from RX
Discard Packet
(b)

Figure 5.34: Helix integration LRBF strategy data packet reception in (a) an OBU and (b) an RSU flowcharts

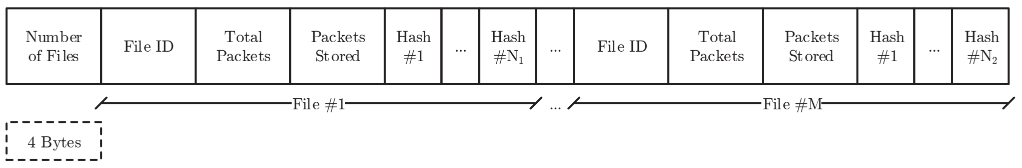


Figure 5.35: HandlerLRBF advertisement packet structure

packet needs to be built using the `peekMainListHashes` method. This method starts to get all the known files ID using the `getAllFilesID` method and recording the number of files known in the return vector. If the node knows any file, it will evaluate which is the known content for each one of them, otherwise the procedure ends here. Thus, for each known file, the file ID, total number of packets of file, and number of packets stored of this file, are recorded in the return vector. If the file is completely downloaded, no more information about that file will be recorded since the neighbors just have to know that this file is completely downloaded in this node. On the other hand, if the file is not completely downloaded, for each element of `mainLinkList` that is associated with this file ID, the corresponding hash in return vector is recorded.

At the end of the procedure the return vector must have the same structure as illustrated in Figure 5.35. Once the information is collected, the header of the advertisement packet is filled as just as the payload (where the collected information is stored). After that, the packet is sent through broadcast and logging variables which count the number of control packet advertised per timestamp(`control_packets_number_per_timestamp`) incremented and the size of the control packets forwarded in this timestamp (`control_packets_size_per_timestamp`) is updated.

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

Reception

The procedure carried out when an advertisement packet is received from the RX module is illustrated in Figure 5.37 and it does not depend on the receiver node's type. So, if the advertisement packet is from a WAVE interface, the payload is isolated from the header and its first 4 bytes (which represent the number of files known by the sender node) are peeked. If the peeked value is greater than zero, all the advertised files will be evaluated. So, the payload content is iteratively analyzed and the file identifier, total number of packets of it, and the number of packets of the file stored by sender node are collected. If the file is completed in the sender node and no information exists about it in the `mapFiles` table, this file is added to `mapFiles` and since the node became aware of a new file to download it should start sending advertisement packets, whereby `is2sendAdv` flag is enabled.

On the other hand, if a file is not completed in the sender node and there are announced hashes to evaluate, the hash is peeked, `vHashes` vector is updated with this hash, and if it is not already in internal structures and in storage, a `Node` is inserted using the `inserNode` method. Otherwise, the `Node` is updated, which means that `listPairs` is updated that can result in a change of its size. If this change happens, the `mainLinkList` must be sorted. At

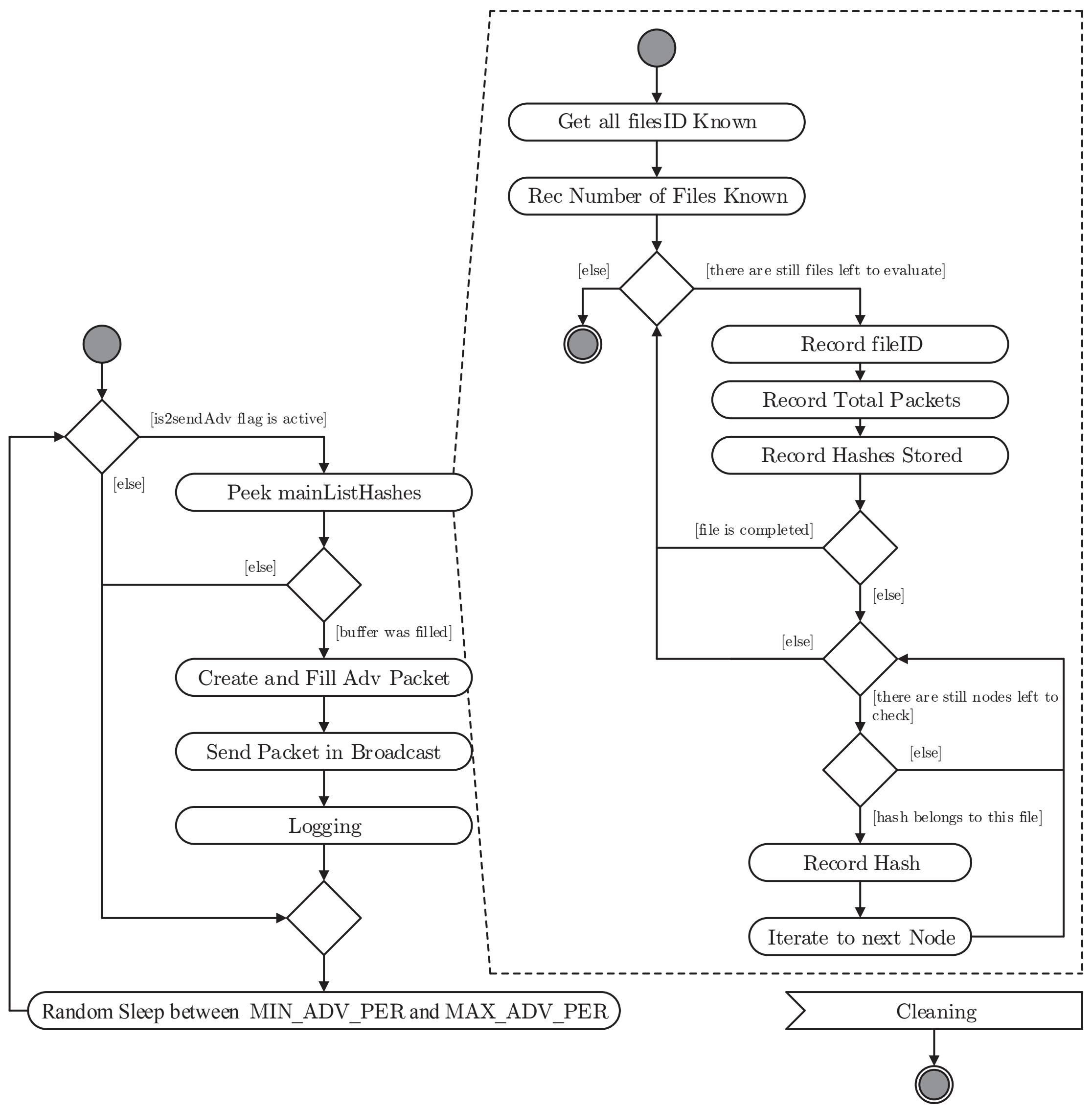


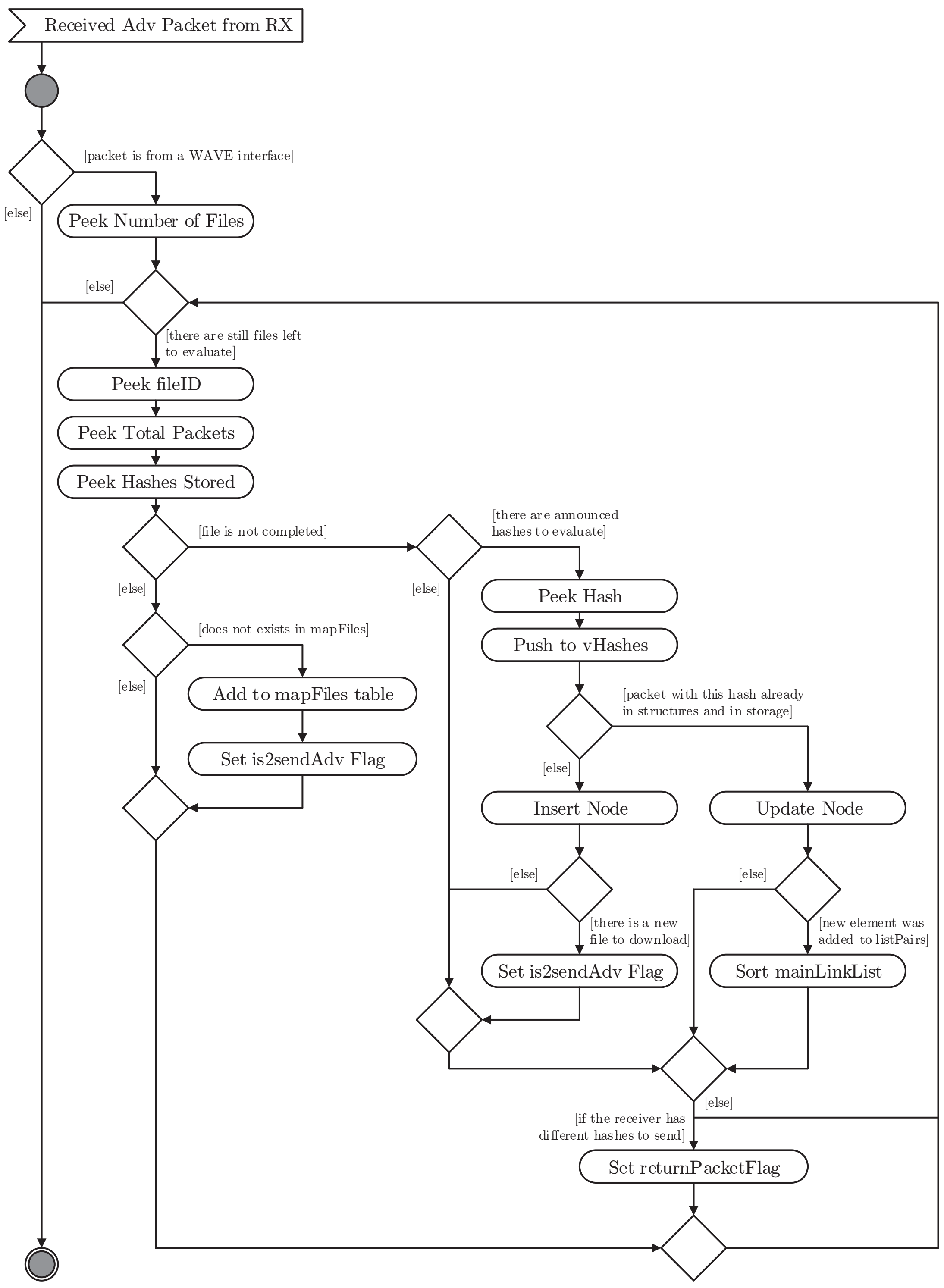
Figure 5.36: Helix integration LRBF strategy advertisement packet forwarding decision flowchart

the end, if the node has any different hashes in its storage, the `returnPacketFlag` must be set to true since there is information that a neighbor does not have all the content. This evaluation is performed by the `hasDifHashesToSend` method.

The previous procedure is repeated until all the advertised files are evaluated.

Initial Generation of Content Distribution Packets

The procedure to initially store packets in RSUs is illustrated in Figure 5.38. Once the number of packets is defined, all the packets are created and stored. First the packet's header is initialized and a packet is created (join of header and payload). After that, the packet is



Received Adv Packet from RX
[packet is from a WAVE interface]
[else]
Peek Number of Files
[else]
[there are still files left to evaluate]
Peek fileID
Peek Total Packets
Peek Hashes Stored
[file is not completed]
[there are announced hashes to evaluate]
[else]
[else]
Peek Hash
Push to vHashes
[does not exists in mapFiles]
[else]
Add to mapFiles table
Set is2sendAdv Flag
[packet with this hash already in structures and in storage]
[else]
Insert Node
Update Node
[else]
[else]
[there is a new file to download]
[new element was added to listPairs]
Set is2sendAdv Flag
Sort mainLinkList
[else]
[if the receiver has different hashes to send]
Set returnPacketFlag

Figure 5.37: Helix integration LRBF strategy advertisement packet reception flowchart

pushed to storage and an empty `Node` in internal structures is created. The `insertEmptyNode` is used since the packet is not from any OBU but it is only internally created. Since a new packet was pushed, also `mapFiles` and `mainLinkList` are updated.

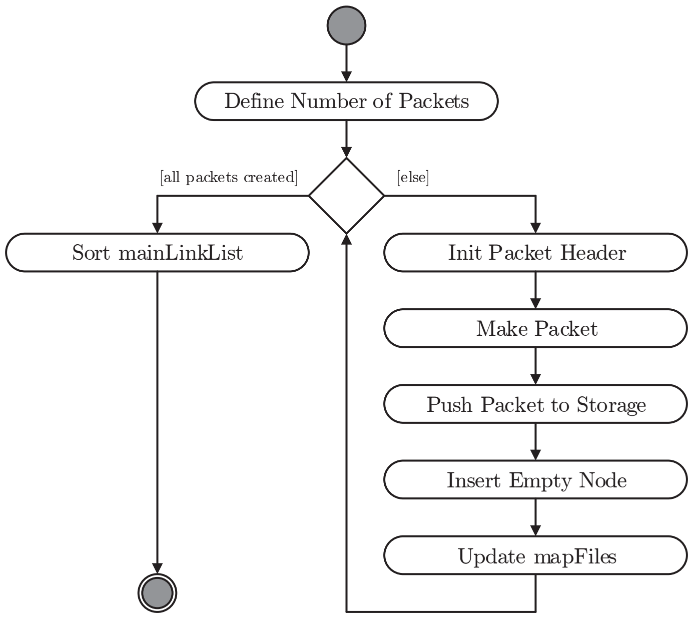


Figure 5.38: Helix integration LRBF strategy generation of content distribution packets flowchart

5.4.4.4 Local Rarest Generation First (LRGF)

In this subsection the implementation of the LRGF strategy in the Helix software is described. There are several differences between this implementation and the last one (of the LRBF strategy). Those differences are justified mainly by the use of the network coding concept in the LRGF content distribution scheme, which is explained in detail in section 4.3. Although the coding module is not deployed, the proposed strategy considered the existence of coded packets to be forwarded and aims to optimize the algorithm to select which coded bundles should be broadcasted first. Thus, during this integration a set of assumptions have been assumed, such as the existence of coded packets to be forwarded (without a deployed coding module) instead of "normal" packets like in the LRBF generation. The emulation of the coding module is achieved using increased network redundancy through the generation of more coded packets. Another relevant information is about the decoding procedure: the receiver node must be able to understand when it is able to decode a certain block (only able if already has received at least `blocksize` packets). The new structures must allow the easy selection according to the most lacking generation of coded bundles in the sender node. Regarding these concerns and considerations several new data structures have to be developed and introduced during the integration of this content distribution scheme.

The LRGF content distribution strategy was implemented through the development of an auxiliary framework mostly responsibly for the identification of the next packet to be forwarded. As mentioned before (see section 4.3.4), in this strategy each node periodically advertises a rank per generation and this information is used in the forwarding decision. To implement this behavior additional control packets were introduced responsible for advertising the node's ranking to their neighbors. As in LRBF strategy, it is also possible to disseminate multiple files at the same time; however, this feature is not the core of the implementation and is only an add-on.

It is important to remember that, when this strategy is used, the content is divided in blocks which are clusters of packets and, when they are coded, each block has a specific generation of coded packets associated. During the following explanation there is no distinction from block and generation since they are closely in terms of implementation.

Auxiliary Structures

The developed architecture relies mostly on the structures illustrated by Figure 5.39. The first one is a block table (`blockTable`) with two elements, where the key element is an identifier of a block and the data element is a pointer to a region of internal memory where the information of such block is stored. Likewise the previous implementation, the second structure is a double-linked list (called `blockList`) of pointers to regions in the internal memory where objects of `Block` type are stored. Since these structures are responsible to keep a tracking about the sender node storage, their number of entries/elements must be equal to the number of blocks/generations of content stored by a node. The last two structures follow the same idea of `hashTable` and `mainLinkList` in the LRBF strategy. However, they are applied to a different type of object of class `Block` which has information about a specific block/generation of a content.

As in LRBF implementation, a table (`mapFiles`) to map the files that are being spread through the network is also used. However, compared to the previous one, it adds a set of additional information in order to achieve a complete knowledge of dissemination contents. The key element of this table is the file ID, and the data element is composed by a tuple of information which includes the size of the file in packets, the number of stored packets, the file size in blocks, number of blocks known by this node, blocksize, and gensize of the file identified by `fileID`.

Moreover, the `Block` class implements two additional structures to handle the implementation of this strategy. The first one is a double-link list (`listRankNeighs`) of a tuple of values which identifies the neighbor (`neighID`), the ranking (`rank`) of the generation associated with this `Block`, and the time at which this information was collected (`time_last`). Since each block/generation has associated a specific set of packets (each one of them identified by a unique hash), another double-link list was created to easily retrieve a hash of this block/generation and monitor when enough packets of the same generation were collected in order to decoded the associated block. If a neighbor goes away from the node's vicinity, the information in `listRankNeighs` should be discarded to perform a more accurate forwarding decision, whereby `time_last` information is very useful.

Analogously to the previous strategies, the `blockTable` and `blockList` were created mainly to provide a faster access to the region in the internal memory where objects of class `Block` are stored, and to quickly peek the hash which identifies the packet to be picked from the storage in order to be sent. Since the forwarding decision relies on the weight of lacking packets from a certain generation in the sender node's neighbors, `listRankNeighs` provides the necessary framework to organize the ranking information. In addition to that, `listHashes` is useful to store an updated state of stored packets belonging to a specific block/generation, providing a quick retrieve of a hash.

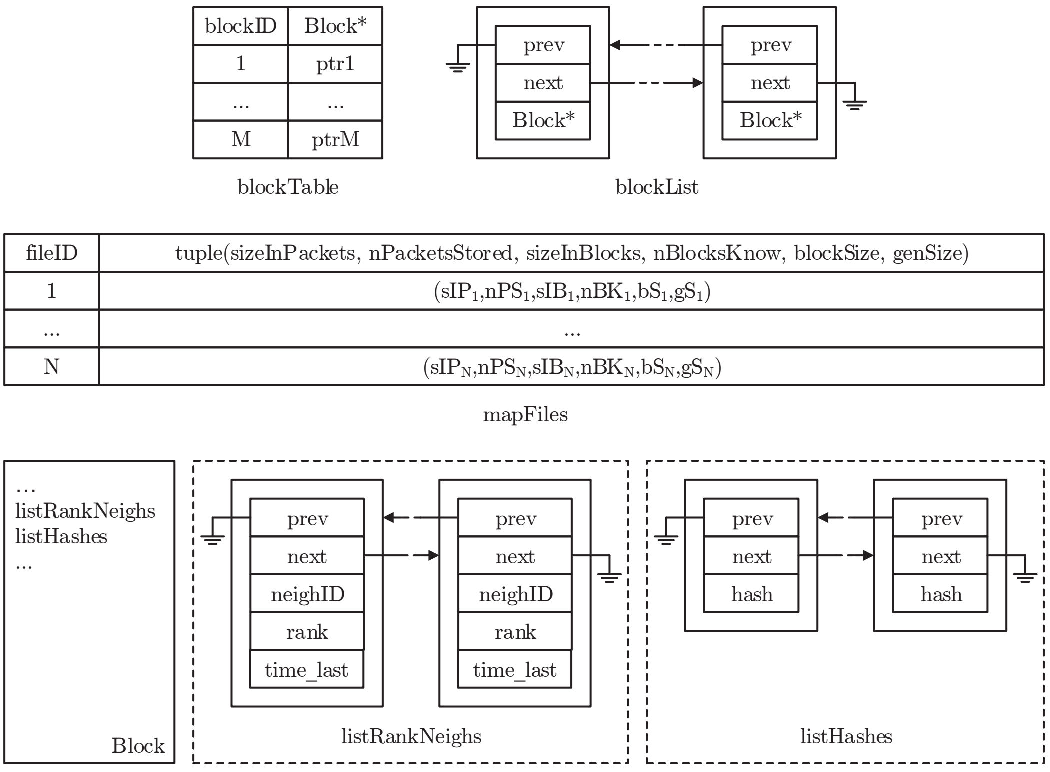


Figure 5.39: Helix integration HandlerLRGF implemented structures

Implementation Scheme

In Figure 5.40 an example of the implemented scheme and how it works concerning the forwarding decision is given. The `blockList` is sorted based on the `sumRank` variable which is a summation of all ranks of the `listRankNeighs` list, and it is updated every time that this list changes. Thus, to prioritize the sending of the most lacking generation, the first element of `blockList` is the `Block` object with a lower `sumRank` although, if there is more than one element with equal values, the number of transmissions is the tie-breaking criteria.

Regarding this previous analysis, it is possible to explain the example of Figure 5.40. The internal memory contains information about four file's blocks (at least some coded packets associated with each one of them was already stored by the sender node) which is organized in objects of type `Block`. The `sumRank` attribute stores the accumulated advertised ranking (by the sender node neighbors) associated with a specific block/generation (identified by `blockID`), which allows the understanding of which is the most lacking generation of coded packets in the sender node vicinity. The `blockList` is a double-link list which allows a faster selection of the next packet to be forwarded (pointed by `head`), whereby it is sorted according to `sumRank`. Since the `Block` associated with the block number `4` is the most lacking generation (smaller value of `sumRank`), a pointer to it is stored in the first position of `mainLinkList` list. Following the same reasoning, the next pointed `Block` is the one identified by `BlockID 1` and pointed by `ptr1`. Since the number of transmission is the tiebreaker criteria, the next coded packet to be sent is the one pointed by `ptr9` and the last one pointed by `ptr6`.

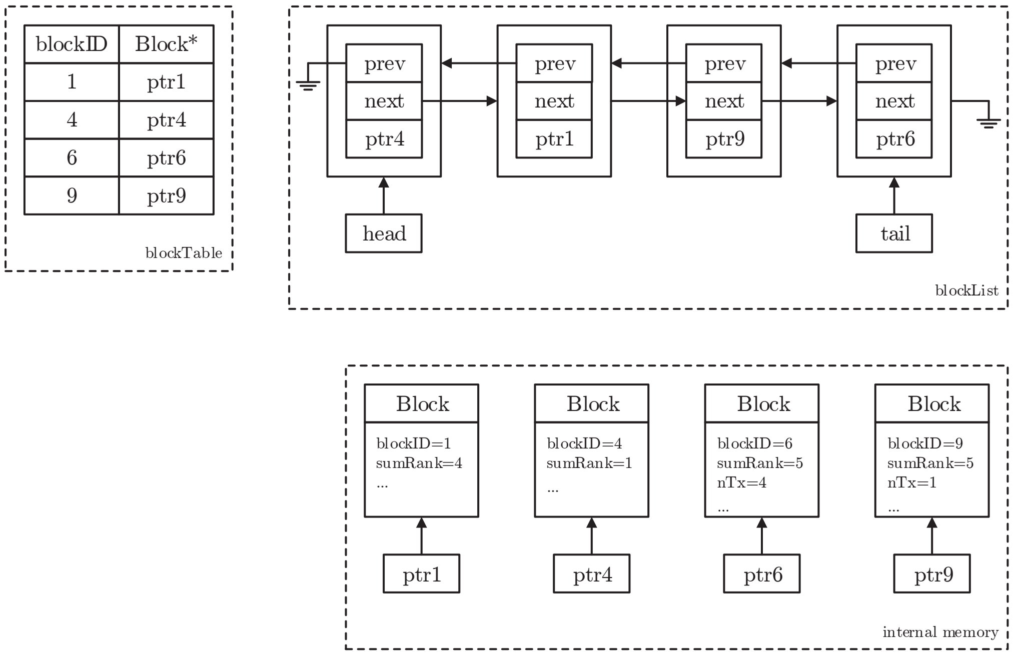


Figure 5.40: Helix integration HandlerLRGF implementation scheme

Implemented Code

Similar to the previous strategies, a new class of objects to handling the implementation of LRGF strategy called `HandlerLRGF` was developed. Its content (attributes and methods) is described in Figure 5.41. Thus, the Routing module creates an entity of this class, called handlerLRGF to handle the forwarding decision in this content distribution strategy. Moreover, and likewise the previous handling classes, it implements an inner class called Block. In the LRGF strategy the packets are organized according to their block or, if they are coded packets, their generation, whereby each object of class Block has a unique block/generation associated, where multiple hashes can be referred.

Block's class has several attributes that are used to perform forwarding decision according to the LRGF strategy. The most relevant are: fileID, blockID, sumRank, and nTx. All the other attributes are also crucial to the implementation, although they are not directly used as a criteria in the forwarding decision. A set of thread-safe methods to access and manage these attributes was developed. There are some which perform key operations during the handling of this strategy, such as incnTx (to increment the number of transmissions of a certain block), updateSumRank (used to update the sumRank attribute which is the key sorted parameter to the blockList), getSumRank, updateListRankNeighs (used when information of an already known neighbor is fresher than older data), refreshListRankNeighs (used to check the validation of the listRankNeighs)), and listRankNeighsComp.

The listRankneighs is updated adding a new element or, if a certain neigh_id already exists, updating the time_last parameter. To do this, the updateListRankNeighs method is used. Since the elements of listRankNeighs have a limited lifetime, the refreshListRankNeighs method is periodically called to check the list's content validity. Every element that was not updated for a period of ELEMENT_VALID_TIME seconds, is deleted from the list.

The blockList is sorted according to the sender node neighbors' ranking. This information is stored in listRankNeighs, and for each element there is a corresponding element with a specific rank. The sorting procedure is done using a sum of all element ranks within the same Block. Thus, the getSumRank and updateSumRank are used to collect and update this information. The sumRank attribute must be updated every time a modification in the listRankNeighs is performed.

Finally, listRankNeighsComp is used as a comparison function of a built-in sorted function of the C++ *list* template when listRankNeighs is sorted. This method sets as the first element of listRankNeighs the one which has been updated the longest. This approach is a key feature regarding a faster update of this list.

Moreover, the class provides two different constructors to allow the creation of new Block objects in one of two ways: (i) an object with an empty listRankNeighs, or (ii) with one element within it. Another list, called listHashes, was created and it is crucial to select what are the returned hashes at forwarding decision moment. Within it are stored all the packet's hashes of this block/generation (identified by blockID) which a node has in its storage. This list of information is used to quickly and accurately return a valid packet hash (hash associated with a stored packet).

In order to implement the HandlerLRGF class several well-known templates of C++ were used. Such as *map*, which is used to create the mapFiles and blockTable structures, and also the *list* template, used in blocklist. The HandlerLRGF class is the main entity in the whole handling procedure, since it is responsible for ensuring the communication between objects of this class and inner objects of the Block class, and also to manage the access and manipulation of the internal memory.

A large and diverse set of thread-safe methods were developed to provide external access and manipulation of Block objects, but also to enable the internal communication between members of HandlerLRGF class and objects of its inner class Block. Among them the most relevant are peekBlockListIDsRank, add2BlockList, add2BlockTable, add2MapFiles, runRefresh, sortBlockList. As mentioned before, Block is an inner class of HandlerLRGF, whereby the methods to access and manage its internal member from outside are implement in the outer class. Among these, the insertBlock, insertEmptyBlock, getFirstAvailableHash, removeBlock, updateBlock, incnTxBlock, and getBlock are noteworthy. A deeper explanation of these methods will be given along the document as required.

As aforementioned all the methods are thread-safe, therefore it is possible to access and manage the internal structures (blockTable, mapFiles, and blockList) at the same time, since they have an independent mutex. However, each object of class Block can only be accessed by one entity at a time.

Start and End of Dissemination

Similarly to the LRBF strategy two auxiliary flags are used to define when a node can send data and/or advertisement packets, returnPacketFlag and is2sendAdv, respectively. The is2sendAdv is initialized as true in RSUs and false in OBUs. If this flag is enabled, a node can broadcast advertisement packets to its vicinity, otherwise it cannot. The other one, returnPacketFlag, is initialized as false in both RSUs and OBUs and, when enabled, allows the broadcast of data packets.

At the beginning, any node sends data packets and only RSUs send advertisement packets. The procedure to enable the broadcast of data packets is described in Figure 5.42-(a). Thus, the node which has enabled the is2sendAdv flag, periodically sends advertisement packets announcing its storage content (how many and which are the blocks that it has, along with the associated ranking of each one of them). When an OBU which does not have any content yet receives this advertisement, it updates the internal structures and changes its is2sendAdv flag to true since it gets to know that there is content to be downloaded. After that it announces that it does not have any packet of such content. When node A receives this advertisement, it acknowledges that there is at least one node within its vicinity that does not have any data associated with the content under dissemination. In addition to that, if the advertisement packet reports an incomplete content and the node B has data, it starts broadcasting data packets, whereby its returnPacketFlag is changed to true.

Contrarily, as shown in Figure 5.42-(b), a node should end the dissemination process when it does not have any neighbors or if its vicinity has completed the download. When these conditions are reached, the refresh thread empties the current information about the node vicinity (empties listRankNeighs) and disables the broadcasting of data packets (sets the returnPacketFlag to false).

Refresh Thread

The information collected from the advertisement packets has a limited lifetime of ELEMENT_-VALID_TIME seconds. Thus, a thread to periodically check the validity of listRankNeighs' elements and erase the expired ones was implemented.

The operational procedure of this thread is described in Figure 5.43 which is close to the equivalent implemented thread for the LRBF strategy. Therefore, the process starts with the collection of the actual time (time_ref). After that, and until all elements of blockList have been evaluated, each one of the listRankNeighs is updated through the refreshListRankNeighs method. This method iterates through all the elements of listRankNeighs until it reaches a valid one. When an expired element is detected, it is erased from the list and the next one is checked. However, if a valid one is detected, all further elements are also valid since the listRankNeighs is sorted according to the timestamp when each list entry was

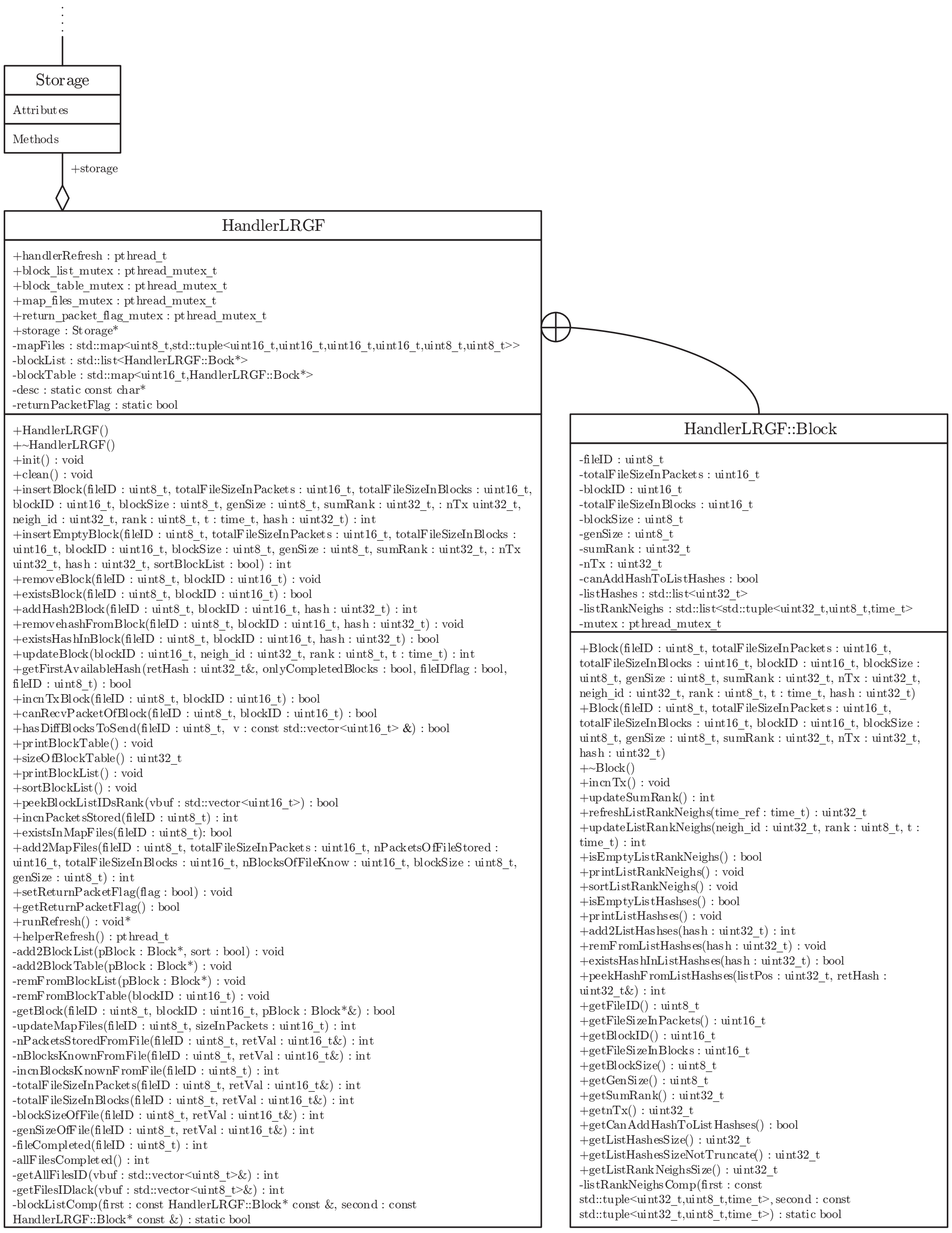

Storage
Attributes
Methods
+storage
HandlerLRGF
+handlerRefresh : pthread_t
+block_list_mutex : pthread_mutex_t
+block_table_mutex : pthread_mutex_t
+map_files_mutex : pthread_mutex_t
+return_packet_flag_mutex : pthread_mutex_t
+storage : Storage*
-mapFiles : std::map<uint8_t,std::tuple<uint16_t,uint16_t,uint16_t,uint16_t,uint8_t,uint8_t>>
-blockList : std::list<HandlerLRGF::Bock*>
-blockTable : std::map<uint16_t,HandlerLRGF::Bock*>
-desc : static const char*
-returnPacketFlag : static bool
+HandlerLRGF()
+~HandlerLRGF()
+init() : void
+clean() : void
+insertBlock(fileID : uint8_t, totalFileSizeInPackets : uint16_t, totalFileSizeInBlocks : uint16_t, blockID : uint16_t, blockSize : uint8_t, genSize : uint8_t, sumRank : uint32_t, : nTx uint32_t, neigh_id : uint32_t, rank : uint8_t, t : time_t, hash : uint32_t) : int
+insertEmptyBlock(fileID : uint8_t, totalFileSizeInPackets : uint16_t, totalFileSizeInBlocks : uint16_t, blockID : uint16_t, blockSize : uint8_t, genSize : uint8_t, sumRank : uint32_t, : nTx uint32_t, hash : uint32_t, sortBlockList : bool) : int
+removeBlock(fileID : uint8_t, blockID : uint16_t) : void
+existsBlock(fileID : uint8_t, blockID : uint16_t) : bool
+addHash2Block(fileID : uint8_t, blockID : uint16_t, hash : uint32_t) : int
+removehashFromBlock(fileID : uint8_t, blockID : uint16_t, hash : uint32_t) : void
+existsHashInBlock(fileID : uint8_t, blockID : uint16_t, hash : uint32_t) : bool
+updateBlock(blockID : uint16_t, neigh_id : uint32_t, rank : uint8_t, t : time_t) : int
+getFirstAvailableHash(retHash : uint32_t&, onlyCompletedBlocks : bool, fileIDflag : bool, fileID : uint8_t) : bool
+incnTxBlock(fileID : uint8_t, blockID : uint16_t) : bool
+canRecvPacketOfBlock(fileID : uint8_t, blockID : uint16_t) : bool
+hasDiffBlocksToSend(fileID : uint8_t, v : const std::vector<uint16_t> &) : bool
+printBlockTable() : void
+sizeOfBlockTable() : uint32_t
+printBlockList() : void
+sortBlockList() : void
+peekBlockListIDsRank(vbuf : std::vector<uint16_t>) : bool
+incnPacketsStored(fileID : uint8_t) : int
+existsInMapFiles(fileID : uint8_t): bool
+add2MapFiles(fileID : uint8_t, totalFileSizeInPackets : uint16_t, nPacketsOfFileStored : uint16_t, totalFileSizeInBlocks : uint16_t, nBlocksOfFileKnow : uint16_t, blockSize : uint8_t, genSize : uint8_t) : int
+setReturnPacketFlag(flag : bool) : void
+getReturnPacketFlag() : bool
+runRefresh() : void*
+helperRefresh() : pthread_t
-add2BlockList(pBlock : Block*, sort : bool) : void
-add2BlockTable(pBlock : Block*) : void
-remFromBlockList(pBlock : Block*) : void
-remFromBlockTable(blockID : uint16_t) : void
-getBlock(fileID : uint8_t, blockID : uint16_t, pBlock : Block*&) : bool
-updateMapFiles(fileID : uint8_t, sizeInPackets : uint16_t) : int
-nPacketsStoredFromFile(fileID : uint8_t, retVal : uint16_t&) : int
-nBlocksKnownFromFile(fileID : uint8_t, retVal : uint16_t&) : int
-incnBlocksKnownFromFile(fileID : uint8_t) : int
-totalFileSizeInPackets(fileID : uint8_t, retVal : uint16_t&) : int
-totalFileSizeInBlocks(fileID : uint8_t, retVal : uint16_t&) : int
-blockSizeOfFile(fileID : uint8_t, retVal : uint16_t&) : int
-genSizeOfFile(fileID : uint8_t, retVal : uint16_t&) : int
-fileCompleted(fileID : uint8_t) : int
-allFilesCompleted() : int
-getAllFilesID(vbuf : std::vector<uint8_t>&) : int
-getFilesIDlack(vbuf : std::vector<uint8_t>&) : int
-blockListComp(first : const HandlerLRGF::Block* const &, second : const HandlerLRGF::Block* const &) : static bool
HandlerLRGF::Block
-fileID : uint8_t
-totalFileSizeInPackets : uint16_t
-blockID : uint16_t
-totalFileSizeInBlocks : uint16_t
-blockSize : uint8_t
-genSize : uint8_t
-sumRank : uint32_t
-nTx : uint32_t
-canAddHashToListHashes : bool
-listHashes : std::list<uint32_t>
-listRankNeighs : std::list<std::tuple<uint32_t,uint8_t,time_t>
-mutex : pthread_mutex_t
+Block(fileID : uint8_t, totalFileSizeInPackets : uint16_t, totalFileSizeInBlocks : uint16_t, blockID : uint16_t, blockSize : uint8_t, genSize : uint8_t, sumRank : uint32_t, nTx : uint32_t, neigh_id : uint32_t, rank : uint8_t, t : time_t, hash : uint32_t)
+Block(fileID : uint8_t, totalFileSizeInPackets : uint16_t, totalFileSizeInBlocks : uint16_t, blockID : uint16_t, blockSize : uint8_t, genSize : uint8_t, sumRank : uint32_t, nTx : uint32_t, hash : uint32_t)
+~Block()
+incnTx() : void
+updateSumRank() : int
+refreshListRankNeighs(time_ref : time_t) : uint32_t
+updateListRankNeighs(neigh_id : uint32_t, rank : uint8_t, t : time_t) : int
+isEmptyListRankNeighs() : bool
+printListRankNeighs() : void
+sortListRankNeighs() : void
+isEmptyListHashses() : bool
+printListHashses() : void
+add2ListHashses(hash : uint32_t) : int
+remFromListHashses(hash : uint32_t) : void
+existsHashInListHashses(hash : uint32_t) : bool
+peekHashFromListHashses(listPos : uint32_t, retHash : uint32_t&) : int
+getFileID() : uint8_t
+getFileSizeInPackets() : uint16_t
+getBlockID() : uint16_t
+getFileSizeInBlocks : uint16_t
+getBlockSize() : uint8_t
+getGenSize() : uint8_t
+getSumRank() : uint32_t
+getnTx() : uint32_t
+getCanAddHashToListHashses() : bool
+getListHashesSize() : uint32_t
+getListHashesSizeNotTruncate() : uint32_t
+getListRankNeighsSize() : uint32_t
-listRankNeighsComp(first : const std::tuple<uint32_t,uint8_t,time_t>, second : const std::tuple<uint32_t,uint8_t,time_t>) : static bool

Figure 5.41: Helix integration HandlerLRGF sub-module collaboration diagram

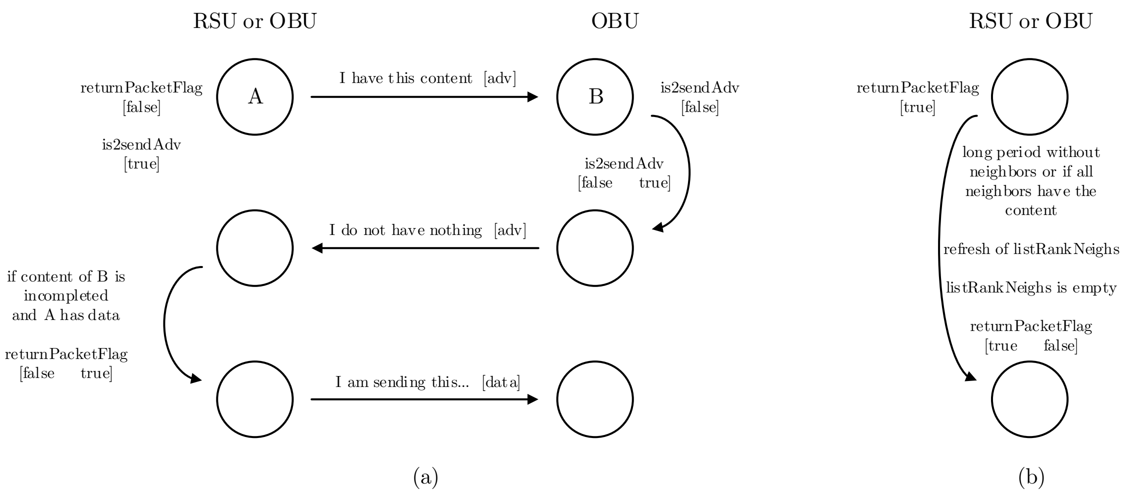

RSU or OBU
OBU
RSU or OBU
returnPacketFlag [false]
A
I have this content [adv]
B
is2sendAdv [false]
is2sendAdv [true]
is2sendAdv [false true]
I do not have nothing [adv]
if content of B is incompleted and A has data
returnPacketFlag [false true]
I am sending this... [data]
(a)
returnPacketFlag [true]
long period without neighbors or if all neighbors have the content
refresh of listRankNeighs
listRankNeighs is empty
returnPacketFlag [true false]
(b)

Figure 5.42: Helix integration HandlerLRGF (a) start and (b) end of dissemination

updated (`time_last` value), being the first element the oldest one. An element is considered to be valid if it was updated in the previous period of `ELEMENT_VALID_TIME` seconds (or until `time_ref-time_last` is lower than `ELEMENT_VALID_TIME`).

If in the refreshing process any element was erased from `listRankNeighs`, this structure needs to be sorted (flag `is2sort` is set to true) and the `sumRank` attribute (responsible for keeping a record about the ranking of a specific generation in the node vicinity) is updated, its value must change in order to be in compliance with the `listRankNeighs` content. Furthermore, if the node does not have information about its vicinity ranking (since `listRankNeighs` was empty), the `isAllEmpty` flag is enabled and the node stops the broadcasting of coded packets (`returnPacketFlag` is disabled) since there is no longer information to perform the forwarding decision. This procedure guarantees the end of the dissemination process when all neighbors have the content or if there are no neighbors for a long period of time. On the other hand, if `listRankNeighs` still has a valid element, the `returnPacketFlag` is enabled. Since changes in `listRankNeighs` were performed, the `blocklist` must be sorted in order to be in compliance during the forwarding decision.

This thread is executed with a random periodicity between `MIN_REFRESH_PERIOD` and `MAX_REFRESH_PERIOD` seconds to avoid possible starvation caused by a fixed periodicity.

Data Packet Forwarding Decision

A similar approach to the previous implementations was taken in order to make a forwarding decision according to the LRGF strategy, and its flowchart operation is illustrated in Figure 5.44. Thus, the `runRouting` thread was modified and is equal in both types of nodes. The thread is awake with a periodicity of `CYCLE_DELAY` microseconds (default value is 200 ms).

When the thread is awake, if the node has neighbors and all of them are OBUs, and its storage is not empty, the internal structures implemented to handle this strategy will be queried in order to evaluate which is the packet to be sent. To perform this operation, the `getFirstAvailableHash` method is used since it is the method developed to return the hash of the coded packet that should be sent first. Within this method, if the broadcasting of data packets is enable (`returnPacketFlag` is true) the taken procedure is as follows.

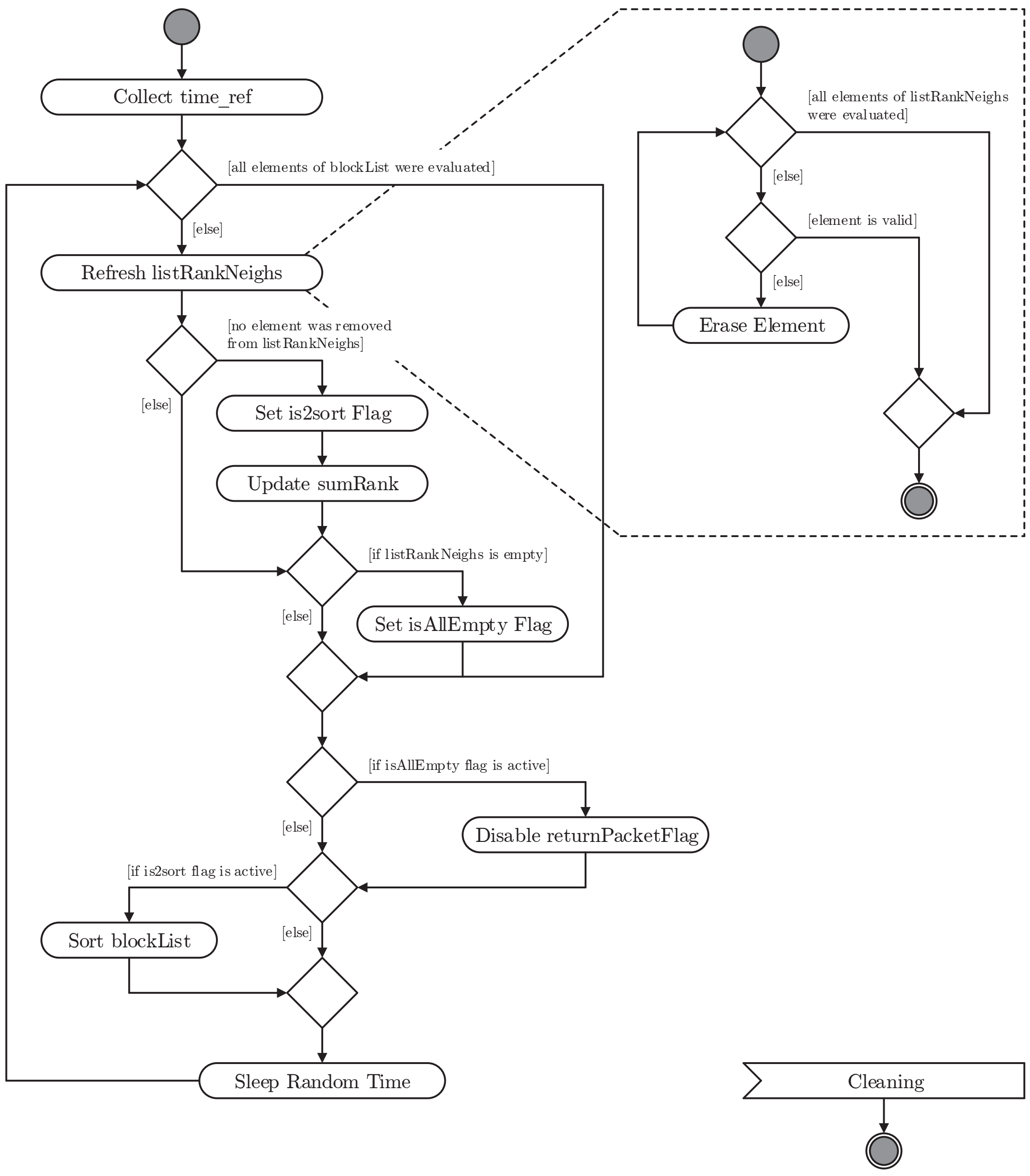


Figure 5.43: Helix integration LRGF strategy refresh thread flowchart

Since the `blockList` is sorted according to the ranking, the first element will be the one with a lower value of `sumRank` (meaning that block/generation is the most lacking one in the node's vicinity). Thus, the `blockList` is iterated until a valid hash from this survey is not returned. There are two ways to perform the survey: (i) only considering packets from completed decoded blocks, or (ii) enabling the return of hashes belonging to packets of a generation that is not decoded yet (meaning that this packet was coded by another node and this node will only forward it). These two situations are distinguished through the setting of completeGenFlag; when enabled only coded packets from generations associated with completed decoded blocks can be forwarded; otherwise is the second one which is deployed. Thus, the iterative process through blockList continues until a valid Block is reached. After that, the size of the listHashes associated with the selected Block is evaluated in order to randomly generate a number which identifies the position in listHashes where the hash to be returned is stored. This position is evaluated using the information returned by getListHashesSizeNotTruncated method, which returns the listHashes size not truncated to blocksize hashes. Once the hash is peeked, and if it is available in storage, the procedure jumps to its end. Otherwise, it re-evaluates the hash to be peeked or, if there are no more attempts, it iterates to the next Block.

Once the hash of the packet to be pushed from storage is collected and if it is a content distribution packet, its prevEID (identifier of the packet source node) and numNeigh (number of hops) fields are updated (similar to the previous strategies). After that, the packet is sent in broadcast to the nodes's vicinity and (in the internal structures) the number of transmissions of the associated Block is incremented. Since one of the sorting parameters of blockList was updated, it must be sorted using the sortBlockList method. Likewise the previous implementations, at the end of the procedure the logging variables packets_transmitted_-total and packets_transmitted_per_timestamp are incremented.

Finally, the runRouting thread runs until a cleaning signal is sent.

Data Packet Reception

Figure 5.45 and Figure 5.46 describe what are the procedures when a node (OBU or RSU) receives a data packet from the RX module. Similarly to the other implementations, when an RSU receives a data packet, it discards it since the transfer of data information one-directional (from RSUs to OBUs or among OBUs). Otherwise, if the packet is received in an OBU and comes from a WAVE interface, a set of procedures will be performed.

First of all, the packet's hop list is updated with the ID of the receiver node. If the Block associated with the received packet does not exist in the internal structures, and that packet comes from a valid neighbor, a new object of type Block will be inserted.

The creation of this new element depends on the type of the sender node. If the packet is received from an OBU, a completed new Block is added using the insertBlock method. This method allows the creation of a new Block and also initializes the listRankNeighs and listHashes with new elements composed by the sender ID, a default rank (value is set to 0), and, in the case of listHashes, with a new element containing the received packet's hash. This last set of operations is achieved using the Block's constructor composed by 11 input arguments.

If the packet was received from an RSU, it is not relevant to register information about the sender node, since it is not a potential receiver of information in the dissemination process, whereby the internal structures used for handling the forwarding decision do not need to be updated with this kind of information. However, it is important to store information about the received content, whereby an empty Block is created using the insertEmptyNode method. This method creates a new Block (and respectively new entries in internal structures) with an empty listRankNeighs and listHashes. This approach precludes the misrepresenting of metadata crucial to forwarding decisions, and only adds information about a new Block to be downloaded. If new content to download was detected the is2SendAdv flag is enabled in order to trigger the advertisement to further nodes.

On the other hand, if the associated Block already exists, or if procedures finished the

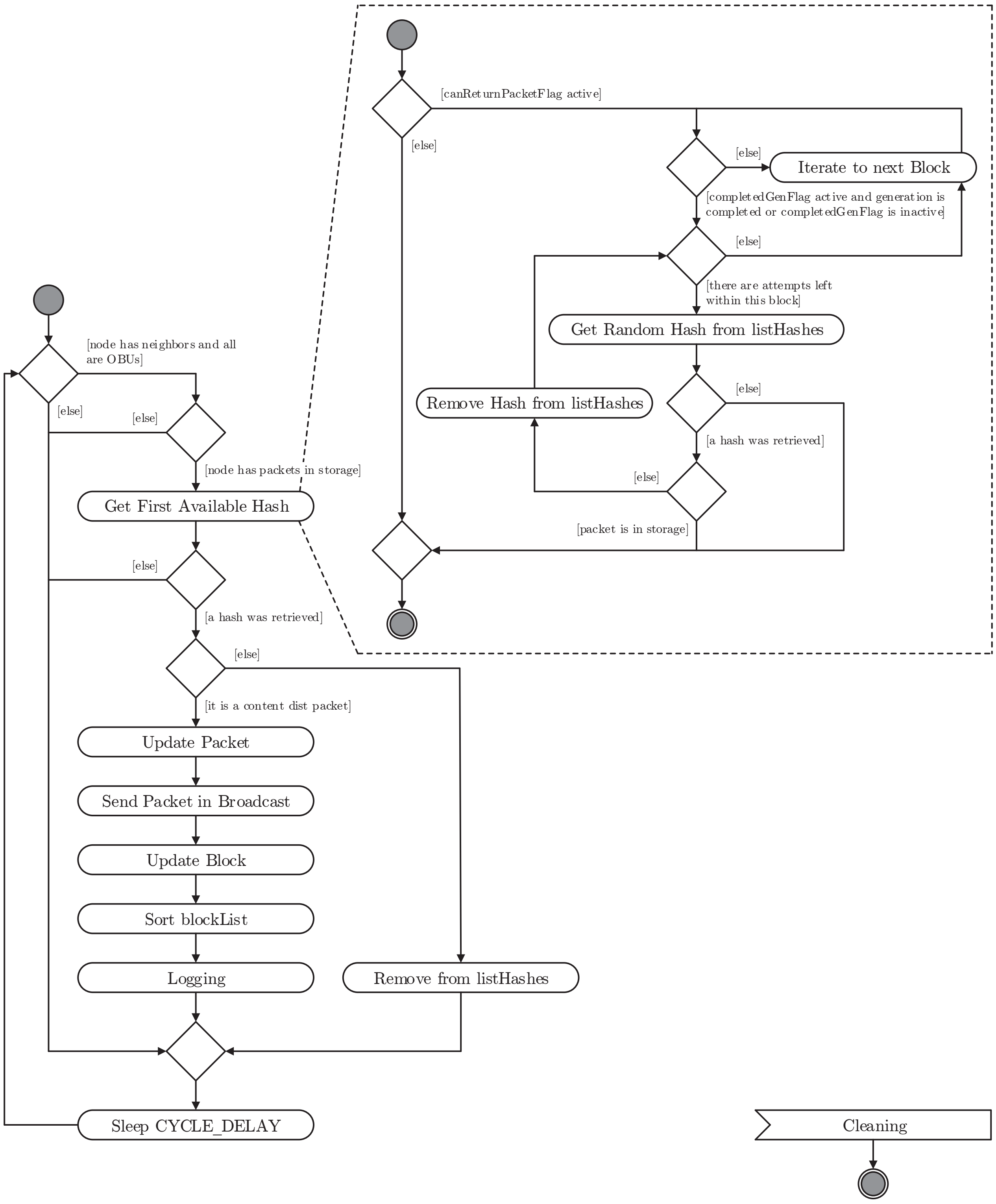

[canReturnPacketFlag active]
[else]
[else]
Iterate to next Block
[completedGenFlag active and generation is completed or completedGenFlag is inactive]
[else]
[there are attempts left within this block]
Get Random Hash from listHashes
[else]
Remove Hash from listHashes
[a hash was retrieved]
[else]
[packet is in storage]
[node has neighbors and all are OBUs]
[else]
[else]
[node has packets in storage]
Get First Available Hash
[else]
[a hash was retrieved]
[else]
[it is a content dist packet]
Update Packet
Send Packet in Broadcast
Update Block
Sort blockList
Logging
Remove from listHashes
Sleep CYCLE_DELAY
Cleaning

Figure 5.44: Helix integration LRGF strategy data packet forwarding decision flowchart

creation of a new one, the node will decide if it can store the received packet or not. Thus, if the node can receive packets of this `blockID` and does not have it in storage, it pushes the packet. A node can only push packets of not completed blocks, whereby a flag was created, called `canAddHashesToListHashes`, per each `Block` in order to signal the authorization (or not) of this procedure. After that, the packet's hash is added to the `listHashes` of the associated `Block` through the `addHash2Block` method which in turn uses the `add2ListHashes`. Thus, if the hash does not exist in `listHashes` it will be added to it. If, with the addition of a new hash, the number of hashes of the some `blockID` (size of `listHashes`) reaches the `blockSize`, `genSize-blockSize` new packets will be generated. This approach enables the emulation of the coding process since it generates the redundancy introduced by this type of strategy. Since a new packet was added to storage, the `mapFiles` table is updated and the number of stored packets is incremented.

Finally, several logging counters are updated, namely, the number of stored and heard packets, along with if the node does not have the received packet the `packets_stored_-total`, `packets_stored_per_timestamp`, and `packets_recv_good_per_timestamp` counters are incremented. Although, if a packet already stored was heard, the `packets_recv_bad_-per_timestamp` is incremented. The `packets_listened_per_timestamp` is incremented in both cases.

Advertisement Procedures

Packet Structure

The implementation of the LRGF strategy mostly relies on advertisement messages which inform which are the blocks/generations and respective ranking a node has. Thus, to implement these messages, packets of advertisement whose structure is described in Figure 5.47 were introduced. This packet has a set of information about the contents that are being spread through the network, but also about the specific content of the sender node storage.

The structure implemented is mainly by two types of information: (i) generic information about a specific file, and (ii) information about the sender node storage content. The first set has elements which identify the file, and delivery generic information about it: number of total packets and blocks, number of packets stored by the sender node and which blocks/generation he knows, and also which is the block size and generation size of defined to this file. After this generic data, a set of pair of values (block/generation ID, ranking) are given. Each pair is composed by an identifier of the block/generation which the sender node knows and its ranking. The ranking is defined in number of packets between 0 (sender node knows the block/generation but does not have any data of it) and `BLOCKSIZE` (sender node knows the block and already decoded it). This set of information is replicated for each known file and any element of it has a size of 2 Bytes. In the following paragraphs is discussed the advertisement packets forwarding decision. Moreover, is explained how a node fills the advertisement packet and how the information is used by the receiver node.

Forwarding Decision

Figure 5.48 illustrate the implemented forwarding decision flowchart of an advertisement packet. Once the `is2sendAdv` flag is enabled, the `peekBlockListIDsRank` is invoked in order to collect all the `Block`'s IDs which this node has store and the ranking associated with this generation of coded packets.

Thus, this method starts to collect all the IDs of the files that are known by this node

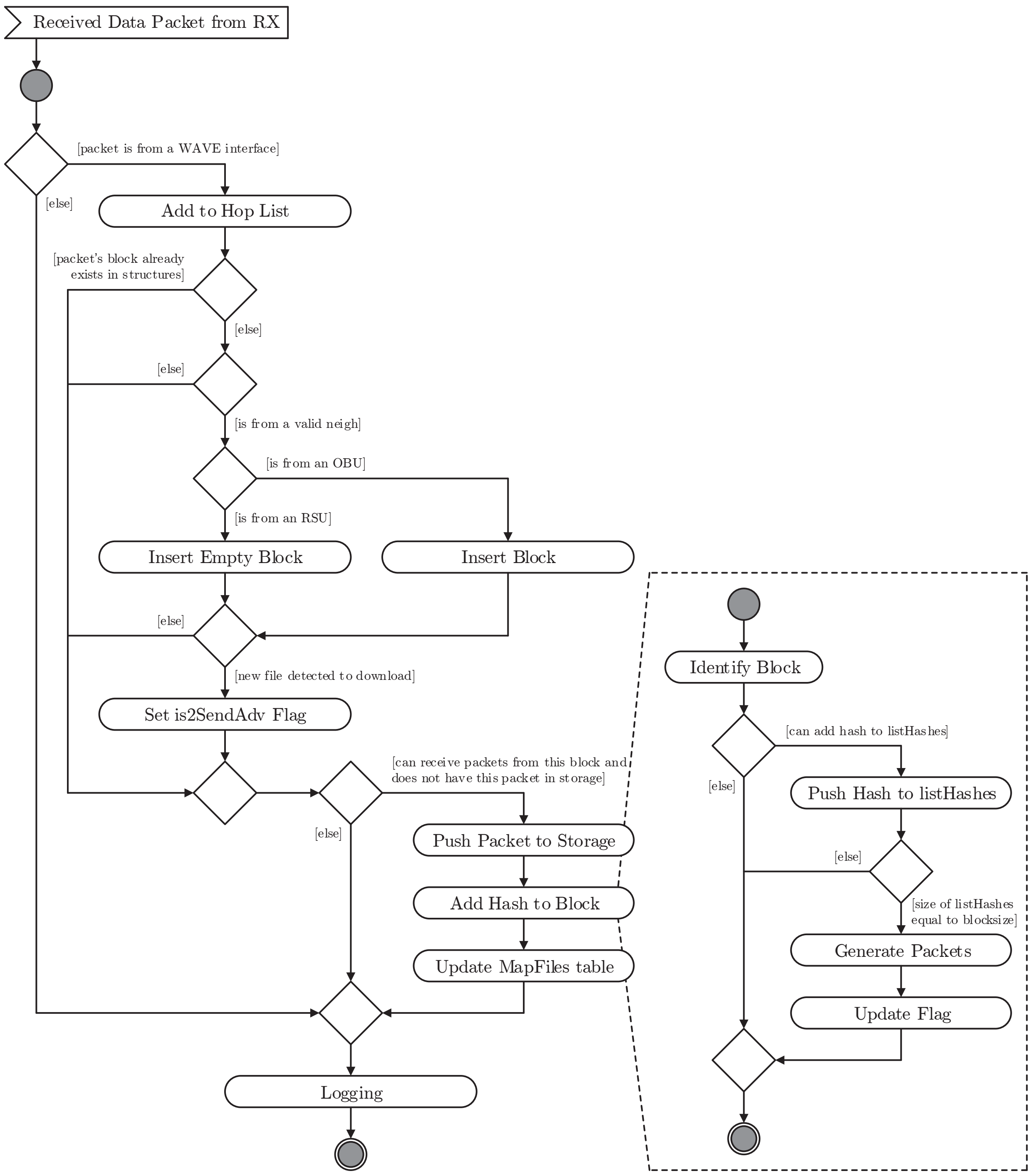


Figure 5.45: Helix integration LRGF strategy data packet reception in an OBU flowchart

(through the `getAllFilesID` method), and records it in the vector to be returned at the end of the procedure. If a node has information about at least one file, for all the collected files are collected and recorded several information about the file. Included in those are the identifier

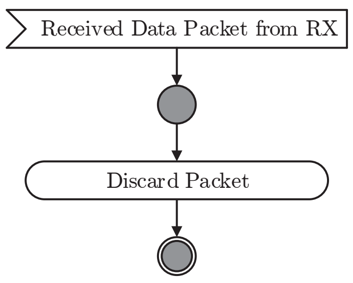


Figure 5.46: Helix integration LRGF strategy data packet reception in an RSU flowchart

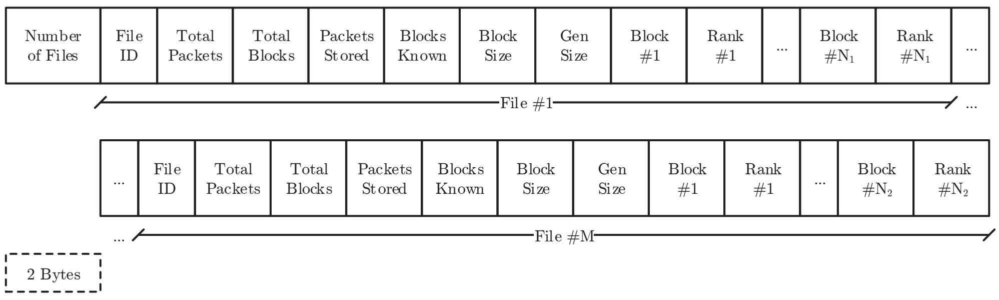


Figure 5.47: HandlerLRGF advertisement packet structure

of the file, the total number of packets and blocks of it, the number of packets stored and blocks known by this node, and also the block and generation size. After this, the procedure fills the return buffer with a set of pair of values representing the stored content of this file. Each pair is composed by the block/generation identifier and the ranking which is the number of coded packets or, if the block it is already decoded, the number of packets (it is equal to `blocksize`). If the node has the complete file, it only sends information about the IDs which identify the files' blocks. However, if the node does not have information about any file, the procedure ends before start all the previous procedure.

Once filled, the buffer passed for `peekBlockListIDsRank`, it must have the same structure as illustrated in Figure 5.47. After that, the header of the advertisement packet is filled and the packet is created joining the header with the payload (where the collected information is stored) and the packet is sent in broadcast. Finally, the logging variables `control_packets_-number_per_timestamp` and `control_packets_size_per_timestamp` are updated.

Similar to the LRBF strategy, this procedure is controlled by a thread (`runAdvertisement`) which is invoked in a random periodical way between `MIN_CONT_ADV_PERIOD` and `MAX_CONT_-ADV_PERIOD`. This procedure runs until a cleaning signal is sent.

Reception

Every time that an advertisement packet is received by a node, it has to execute the procedure illustrated in Figure 5.49. This procedure is invoked in the RX module and it does not dependent on the receiver node's type.

Thus, if the packet comes from a WAVE interface the node peeks the number of files advertised (first 2 bytes). After that, while there are still files left to evaluate, for each one of

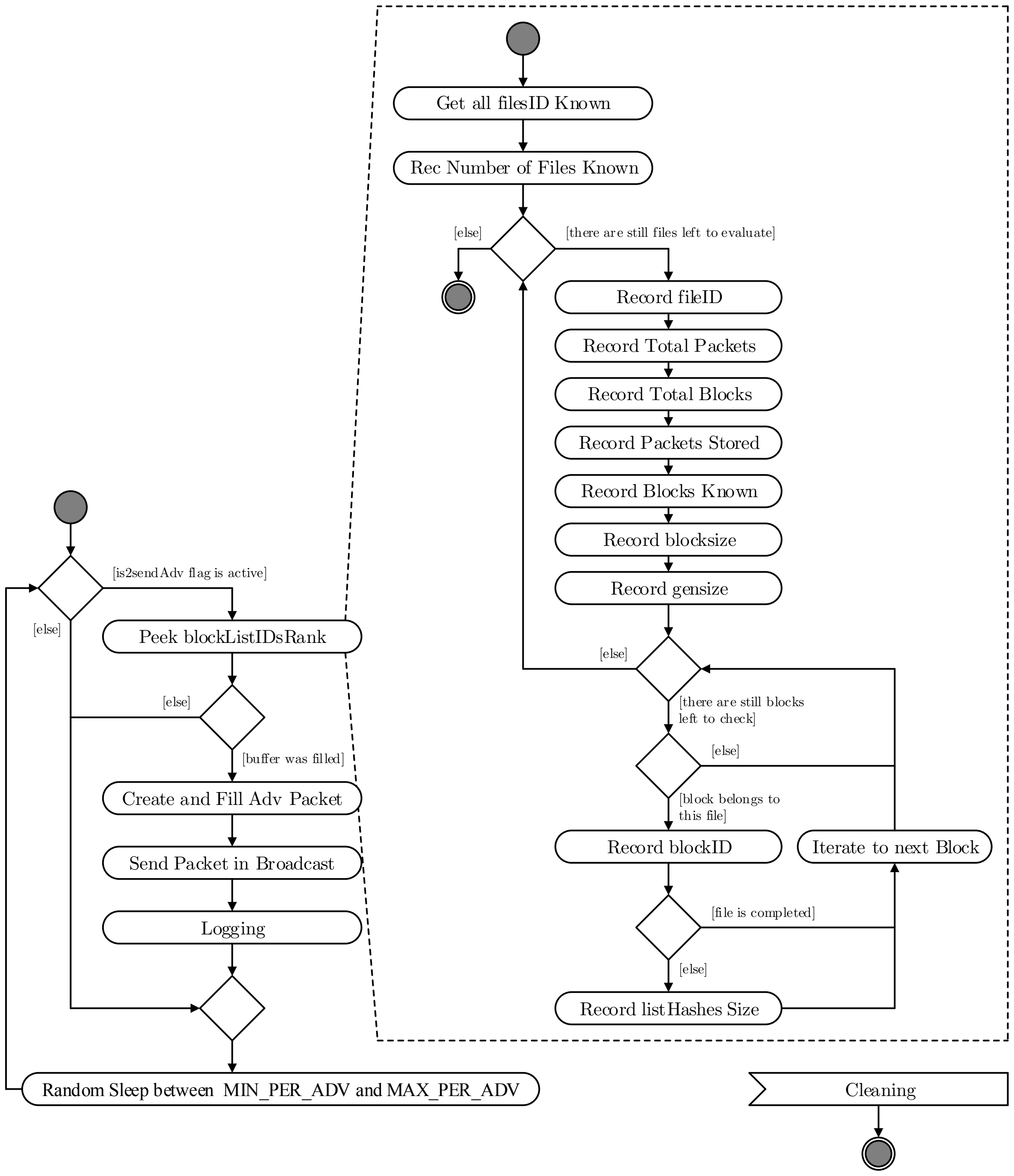


Figure 5.48: Helix integration LRGF strategy advertisement packet forwarding decision flowchart

them it is peeked the identifier of the file, the total number of packets and blocks, the number of packets stored and blocks known by this node, and also the block and generation size.

If the sender node announces that it has the complete file, and such file does not exist in `mapFiles` table of the receiver node, the file is added to the table and the `is2sendAdv` is turned-on, enabling the broadcasting of advertisement packets. As previously mentioned the enabling of this flag is crucial to the beginning of the dissemination process, since the node notice a new file to download.

On the other hand, if the file is not completed, and the sender node announces that it knows (at least) one block from this file, the packet will be analyzed in more detail. Thus, the block ID and rank are peeked, and the `vBlockID` vector is updated with the peek block ID. If this block is not already in the internal structures, a new `Block` is inserted using the `insertBlock` method, and if there is a new file to download the flag `is2sendAdv` is defined as true. However, if already exists a block identified by this `blockID` in the internal structures, the block is updated with this newer information of ranking and, if a new element was added or updated to the `listRankNeighs` variable, the `blockList` is sorted.

At the end, if the receiver has different blockIDs from the same file to send to the sender node the `returnPacket` flag is enabled. Otherwise, the procedure jumps to the beginning of the process in order to analyze the information of the next file.

Initial Generation of Content Distribution Packets

Figure 5.50 illustrates the initial procedure to create and store multiple packets in an RSU in order to be disseminated in the future. This is a very similar process of the one taken into consideration for LRBF strategy. Thus, the header of the packet is created for each one and, after that, a packet is created (attachment of Helix header and payload). Each header is created accordingly to the generation and block that they belong. Once all packets are created, they are pushed to the persistent storage, where they remained during their lifetime. Since the packet is not from any OBU, the `insertEmptyNode` method is used and a new entry in the internal structures is created (along a new `Block`) which is crucial to the dissemination process. Moreover, since a new packet was pushed, also `mapFiles` is updated and `blockList` sorted.

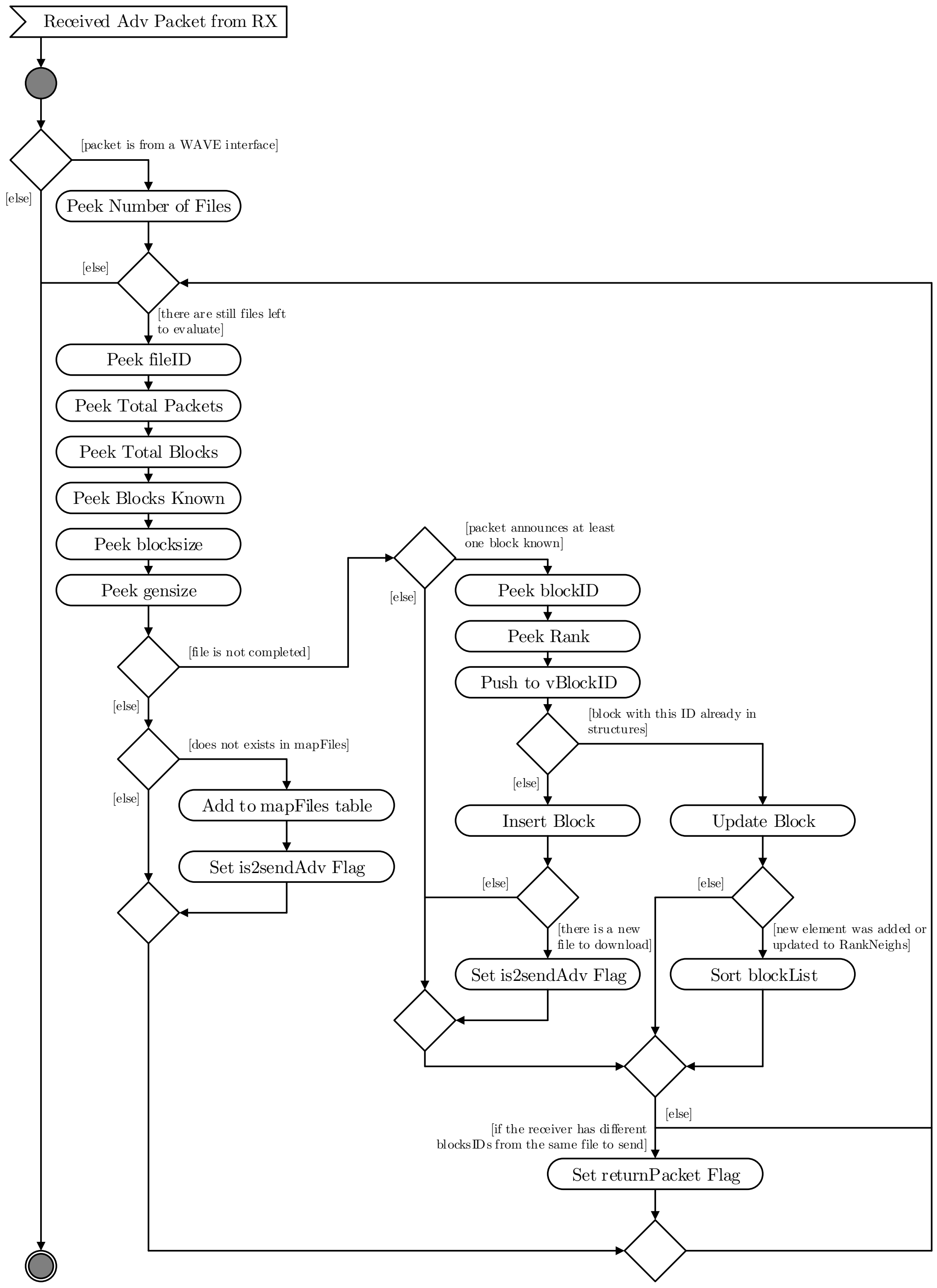
5.5 NetRider Boards Integration

The source code developed for the HelixEmulator is the exact same code that runs on the boards (OBUs or RSUs) of a real vehicular network. When compiled, this code generates a binary file (set of instructions executed by a processor of a computational system). However, the HelixEmulator runs on a linux-based operating system which is different from the one implemented on the boards. The NetRider boards have an OpenWRT-based operating system implemented called VeniamOS[1].

The boards have limited resources (CPU and memory) and the compilation procedure requires a significant amount of time and processing. Moreover, the limited space of the hard disk in the boards does not allow the complete installation of an operating system whereby most of the times several libraries used by the source code of the programs are missing and need to be added to the operating system.

Thus, the source code needs to be cross-compiled from a linux-based system to an OpenWRT-based system (VeniamOS) and a specific processor (AR71xx MIPS32). In order to perform this procedure, the Veniam® has released a build system of the VeniamOS similar to the one of the OpenWRT [160]. This is a set of makefiles and patches that automates the process of

[1]VeniamOS is a proprietary operating system of Veniam® based on OpenWRT [127]



Received Adv Packet from RX
[packet is from a WAVE interface]
[else]
Peek Number of Files
[else]
[there are still files left to evaluate]
Peek fileID
Peek Total Packets
Peek Total Blocks
Peek Blocks Known
Peek blocksize
Peek gensize
[file is not completed]
[else]
[does not exists in mapFiles]
[else]
Add to mapFiles table
Set is2sendAdv Flag
[packet announces at least one block known]
[else]
Peek blockID
Peek Rank
Push to vBlockID
[block with this ID already in structures]
[else]
Insert Block
Update Block
[else]
[there is a new file to download]
Set is2sendAdv Flag
[else]
[new element was added or updated to RankNeighs]
Sort blockList
[else]
[if the receiver has different blocksIDs from the same file to send]
Set returnPacket Flag

Figure 5.49: Helix integration LRGF strategy advertisement packet reception flowchart

---

building a complete VeniamOS-based system that allows the developers to easily generate a cross-compilation toolchain and a root filesystem for embedded systems such as VeniamOS.

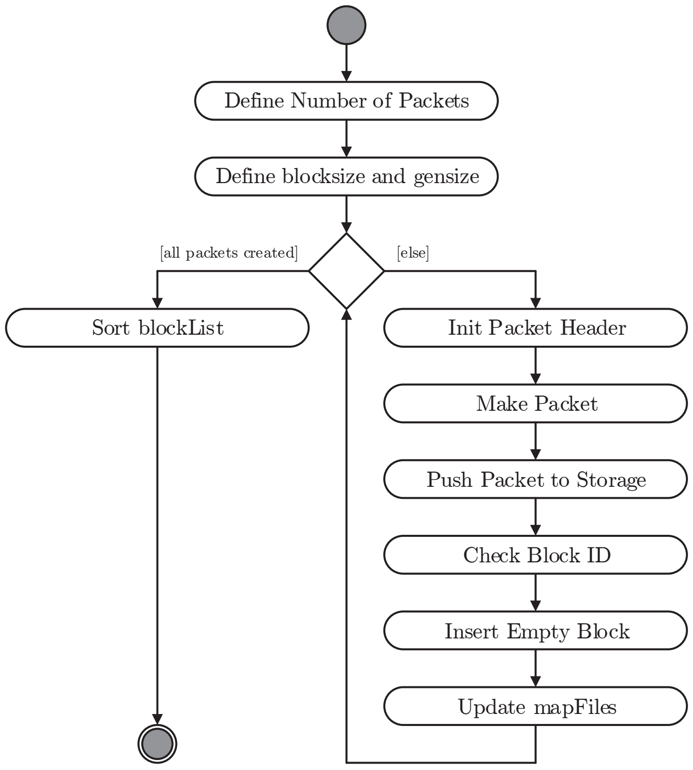


Figure 5.50: Helix integration LRGF strategy generation of content distribution packets flowchart

This toolchain is mainly responsible for generating installer files for the target system which has a different architecture/processor from the one where the build-root is used. Thus, this toolchain was used to cross-compile the code from a computer which has an Intel-64bit system to the boards that use a MIPS32 system.

The VeniamOS build-system was used to cross-compile the source code of two programs. The first one was the Helix support library (called libhelix) and the other was the modified source code of Helix.

Once the source code of the modified Helix is compiled, the program is able to run on the boards. However, this is not enough to perform communication among nodes. The Helix architecture imposed time restrictions for all the nodes in the same network whereby they have to be synchronized to be able to communicate. Thus, this was another key element of the integration in the boards: to provide a correct time synchronization between nodes of the network.

In a real network, the synchronization is assured by the GPS since each node has an equipment of this technology. However, if the network is deployed in a place where there is no GPS signal, other approaches must be taken. Thus, several other synchronization mechanisms were considered such as Precision Time Protocol daemon (PTPd) [161] and Network Time Protocol (NTP) [162].

In the end, the choice resides in the NTP because it is a native protocol of OpenWRT installation. According to Guedes [151], NTP is a networking protocol to synchronize the clocks between computational systems over packet-switched, variable-latency data networks. This protocol aims to synchronize all the devices with Coordinated Universal Time (UTC) assuring a millisecond precision. In order to attenuate the effects of variable network latency and delay, NTP uses an algorithm to select accurate time servers. NTP is usually described as a client-server protocol, but can be easily used in a P2P mode.

This integration results in two documents that aim to handle the installation and running of the developed software in the boards (see Appendix A). The first one is entitled *compileFromHelixEmuToNetRider* and is a set of instructions to compile the source code from the HelixEmu project to the NetRider boards. There are also some specific characteristics of the source code described that were implemented in order to deploy the content distribution schemes on the boards. The second document, called *runContentDistExperimentInBoards* focuses on concerns and warnings regarding the running of a content distribution experiment on the NetRider boards.

5.6 Chapter Considerations

This Chapter focused on the implementation of the content distribution schemes proposed in Chapter 4. A large set of platforms was used to implement and evaluate those strategies. The three platforms covered in this Chapter are a Matlab-based emulator, an emulator specifically designed to develop and perform scalability tests of the DTN mechanisms, and finally the vehicular OBUs used in a laboratory environment. The main considerations of this Chapter are described as follows.

MatlabEmulator

Due to the existence of two large datasets with data collected from a real vehicular network, which has information regarding the vehicles' mobility, position, neighboring, and communication capabilities, a new emulator was developed from the start. The MatlabEmulator was implemented to evaluate content distribution strategies using DTN mechanisms. The mobility model is created using the collected real data. The main reason for its development was the non-existence of an easy and fast platform to develop and evaluate the proposed content distribution strategies without deploying a real network or produce complex code for a typical network emulator. In Chapter 6 this platform is used to evaluate the proposed content distribution schemes. Moreover, during the design process of the proposed strategies, it was used to test and evaluate a variety of approaches and schemes, and proved to be a very useful tool in this process. In the future, it could be used to implement and evaluate other strategies as it is not restricted to the proposed ones.

HelixEmulator and Helix Integration

Although the MatlabEmulator is an efficient tool to design and quickly evaluate content distribution schemes, the code developed on this platform is not prepared to be executed by the DTN software (Helix) used in this work. Thus, a new emulator was developed to allow the implementation of source code directly on this software, whereby the emulated code is the exact code that will be running on the vehicles' boards (when exported to them). Moreover, the emulator has the capability to run hundreds of processes at the same time, enabling the emulation of a large set of vehicles (each process emulates a vehicle), and allowing the testing of the content distribution strategies' scalability. Similarly to the MatlabEmulator, the emulation process is supported by the collected datasets which create a mobility model of the reality. This emulator allows the direct compilation of the source code to the vehicular OBUs, which is a major improvement compared to the MatlabEmulator. In Chapter 6 this platform is used to evaluate the proposed content distribution schemes.

Regarding the integration of the proposed content distribution schemes in the DTN software (Helix), several new modules were created. In this section the implementation and integration of those modules was described in detail. These modules aim to handle the implementation of these strategies. As an example, they implement the internal structures responsible for evaluating which packet should be sent first. Moreover, a Logging module was created in order to collect log information during a content distribution experiment. This module can be used during an emulator experiment and also in a real experiment, and is the source of the logging data used in the evaluation of the proposed schemes in Chapter 6.

NetRider Boards Integration

Finally, in this last section a set of guidelines necessary to integrate and run the developed source code on the NetRider boards was described. Since the OS where the code was developed is not the same as the one installed on the NetRider boards, a cross-compile procedure must be performed. Another challenge is related with the boards' synchronization. In the real network, the boards' synchronization is achieved using GPS technology. However, if the network is deployed in a covered place, whereby there is no GPS signal, the boards have to use P2P or *client-server* synchronization protocols such as NTP or PTPd. As outputs of this integration procedure, two documents were produced regarding the installation and running of the developed software. These two documents describe the majority of the steps performed in the installation and running of the laboratory evaluation in Chapter 6.

Chapter 6

Evaluation

6.1 Chapter Description

Once designed and implemented, the content distribution strategies need to be evaluated in the presented wide range of platforms. Thus, this chapter presents the equipments used in the evaluation, the evaluated scenarios as well as the main results on a variety of platforms.

This chapter is organized as follows:

- *section 6.2 - Equipment and Software*: description of the main characteristics and specification related to the equipment and software used along the wide range of platforms.
- *section 6.3 - Support Scripting*: describes the implemented scripts used to support the evaluation process, to run experiments as well as to perform statistical analysis.
- *section 6.4 - Scenarios and Experiment Description*: detailed description of the scenarios evaluated, covering a set of parameters as number of nodes, geographical area, dissemination periods, among others.
- *section 6.5 - Evaluated Metrics*: presents the metrics used in the statistical analysis in order to evaluate and compare the performance among several content distribution strategies.
- *section 6.6 - Initial Study of the Network*: describes an overview of the network under evaluation based on previously collected log information that allows to obtain a set of global metrics and trends that characterized this vehicular network.
- *section 6.7 - MatlabEmulator Evaluation*: presents the main results achieved through the MatlabEmulator platform.
- *section 6.8 - HelixEmulator Evaluation*: presents the main results achieved through the HelixEmulator platform.
- *section 6.9 - Laboratory Evaluation*: presents the main results achieved through the Laboratory platform.
- *section 6.10 - Chapter Considerations*: depicts the conclusions and the summary of the full chapter.

6.2 Equipment and Software

The following section describes the equipment and software used in the evaluation of the implemented content distribution strategies (on multiple platforms), and also to develop and test the MatlabEmulator and HelixEmulator.

The development of the MatlabEmulator and the evaluation of the proposed scenarios were performed in a machine with the specifications illustrated in Table 6.1. The MATLAB version used in the development is enunciated in Table 6.2, but the emulator is also compatible with newer versions.

Processor	Intel® Core™i7-2670QM CPU @ 2.20 GHz x4
Memory RAM	8.00 GB
Operating System	64-bit Windows 7 Enterprise (SP1)

Table 6.1: Machine used to develop and run the MatlabEmulator

MATLAB®	Version 7.12.0.635 (R2011a)

Table 6.2: Software used to develop and run the MatlabEmulator

The HelixEmulator was developed and modified in one machine which is not the same where it is running. The specifications of this machine are specified in Table 6.3.

Processor	Intel® Core™i5-3230M CPU @ 2.60 GHz x 4
Memory RAM	3.80 GB
Operating System	64-bit Ubuntu 14.04 LTS

Table 6.3: Machine used to develop and modify the HelixEmulator

Due to the heavy computational effort required to run the HelixEmulator and the long emulation time, two identical Virtual Machines (VMs) are used. These VMs were configure using VMware vSphere 5 software and their specifications are described in Table 6.4. Using two VMs enables the emulation of different scenarios in parallel which optimizes the evaluation process. The two VMs were located in different clusters being shared with other processes whereby some experiments may have slightly different values of performance metrics (CPU usage, load, and free memory) when a similar result is expected.

Processor	Intel(R) Xeon(R) CPU E5620 @ 2.40 GHz x 8
Memory RAM	12 GB
Operating System	64-bit Ubuntu 14.04 LTS

Table 6.4: VMs used to run the HelixEmulator

Figure 6.1 shows the boards used in the laboratory experimentation which are the same as used in the FutureCities testbed. This board is an intelligent router that allows the communication among vehicles and between vehicles and infrastructure by using multiple access technologies. Table 6.5 has the main characteristics of the NetRider boards. Moreover, there are a set of additional features that are also relevant:

- Low power consumption (< 6 W).

- IEEE 802.11a/b/g and 802.11p modules.
- High-gain external antennas for each access technology operating in the following frequencies/technologies: 2.4 GHz (IEEE 802.11a/b/g), 5.9 GHz (IEEE 802.11p), and cellular (3G/4G).
- Ethernet, serial (RS-232) and USB ports.

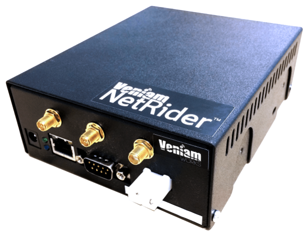


Figure 6.1: NetRider board (RSU/OBU)

Processor	AR71xx MIPS32
Memory RAM	64 MB
Operating System	VeniamOS Release: 05.27.14
Hard Disk	128 MB

Table 6.5: NetRider board specifications

6.3 Support Scripting

In order to support the evaluation of the proposed content distribution strategies several scripts were developed. This set of scripts can be divided into three based on their purpose: (i) update of databases, (ii) support to integration and evaluation in the platforms, and (iii) used for statistical analysis.

In the first group, scripts in MATLAB were developed to generate logging of information of RSUs, since in the collected data of the real-testbed it does not exist (there is only information about OBUs). Thus, using a back-tracking approach based on the logs of OBUs, these logs were created and used as input data in the MATLAB emulator. As an example, if an OBU A has in its list of neighbors (in the real-testbed collected data) the RSU B with an RSSI of X, the generated log of B has a neighbor A with a link's RSSI of X. As in MATLAB, the databases created to be used by the HelixEmulator also did not contain information about RSUs as transmitter nodes. Thus, the created log information for the MatlabEmulator was also loaded in the databases of the HelixEmulator. This loading was achieved by creating a python script which interacts with the already created tables in the MySQL database. Moreover, in one of the datasets used in the evaluation some information was duplicated, whereby a series of amendments have been made in the dataset to handle this situation.

Regarding the integration and evaluation in the multiple platforms used to evaluate the proposed solutions, a set of supporting scripts were developed. The MatlabEmulator runs directly on any type of machine that can run the MATLAB software. On the other hand, the machine where the HelixEmulator was developed and compiled and the machine used to run it are not the same (even though they have a similar operating system). Thus, a bash script was created to automatically compile the source code of the emulator and send the resulting binary files to the running machine. The use of the HelixEmulator involves the creation of two types of processes: master program of the emulator, and several helix programs that could be initiated as OBUs or RSUs. Thus, a bash script was developed in order to launch multiple helix processes and one emulator process stating the node type, the start and the end of the emulation process. For the integration on the NetRider boards, scripts which automatically cross-compile the source code and generate an opkg packet [163] were developed, and installed on the boards. An auxiliary script to run the evaluated experiment was also developed and guarantees an accurate repetition. Moreover, a MATLAB script to evaluate what were the nodes that entered in a specific geographical area during a certain period of time was also developed. This script is crucial to evaluate the content distribution schemes within specific scenarios (e.g. rush hour, parking lot, etc.), and its result is used as an input for both the MatlabEmulator and the HelixEmulator to specify which are the nodes that should be emulated.

Finally, multiple scripts were created to perform the statistical analysis of the large quantity of generated data from the platforms used in the evaluation. The MatlabEmulator already has a built-in module to perform this statistical analysis. On the other hand, since the HelixEmulator and the NetRider boards are not able to directly perform content distribution statistical analysis and collecting of log data, a Logging module was developed. This module produces a large quantity of data that must be analyzed in order to perform an accurate evaluation of the implemented strategies. Moreover, to evaluate the performance (in compliance with subsection 6.5.3) of the machine that runs the HelixEmulator and the NetRider boards a bash script was developed. Thus, a MATLAB script was created to perform a statistical analysis of the data generated by the Logging module and the performance script. In practice, two scripts were developed since one is specific to the HelixEmulator and the other to the NetRider boards. Furthermore, another MATLAB script was developed to facilitate the comparison between the multiple strategies and platforms.

6.4 Scenarios and Experiment Description

6.4.1 OPorto Testbed

In this section a description of the evaluated real testbed is given. Among other characteristics, the context, location, and specifications of this large VANET are described. Moreover, a description of the evaluated scenarios is also given, focusing on different dissemination regions and time periods.

6.4.1.1 Network Overview

This vehicular network located in Oporto city has been deployed in common projects with both Universities of Aveiro and Porto, IT, and Veniam, and is supported by a infrastructure network of the Porto Digital [164]. This testbed interconnects hundreds of vehicles (public buses, garbage trucks, and municipality vehicles) in order to provide a set of services such as Internet access and delay-tolerant communications to transport non-urgent information (sensors and logs).

In order to better understand the network topology and behavior, it has been performed two data collections of 24-hour each. As mentioned in subsection 5.2.6 this data is composed by a variety of information such as GPS coordinates or lists of neighbors, per time sample. The first collection was carried out on the 31st of October, 2014 from 00am to 12pm, and the second one on the 12th and 13th of February, 2015 from 6pm to 6pm. Table 6.6 describes the amount of nodes (OBUs and RSUs) presented in the network during the collection period and also the sampling period of the log data.

Dataset	**OBUs**	**RSUs**	**Sampling Period (T_s)**
October 2014	337	17	5s
February 2015	396	50	2s

Table 6.6: Total number and type of nodes collected for each dataset

From the Table 6.6 an increase in the number of nodes of the network is clearly noticeable. Figure 6.2 and Figure 6.3 visually highlight the enforcement in the road side infrastructures carried out from October 2014 to February 2015. The number of RSUs (red dots) increases more than 190% and the number of OBUs had a lower increase of 17%. Moreover, the coverage area of the network in February is quite larger than in October, extending into the surrounding area of the city center and providing connectivity in one of the most busy roundabout of Oporto (*rotunda da Boavista*).

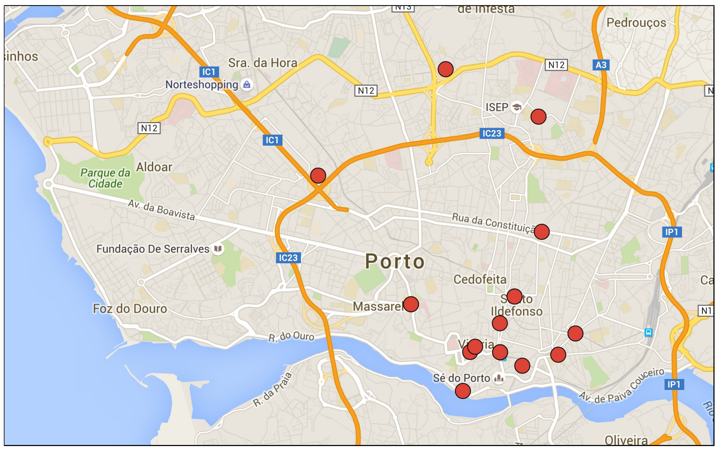


Figure 6.2: Testbed description (October 2014)

6.4.1.2 Evaluated Scenarios

The datasets are composed by data collected from all the network nodes being filtered based on their geographical position or time period. Thus, several scenarios for content distribution can be deployed and evaluated during the 24-hour period where information was gathered. In order to obtain a full understanding about the feasibility of the content distribution schemes, a set of scenarios were evaluated. These scenarios are divided based on geographical regions and time periods as described as follows.

City Center

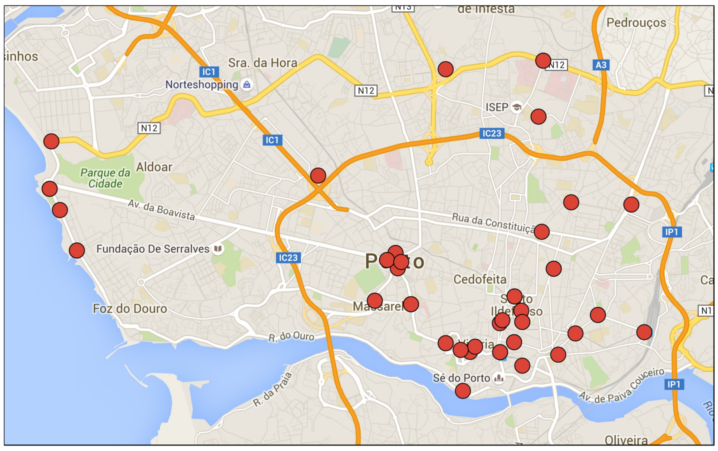
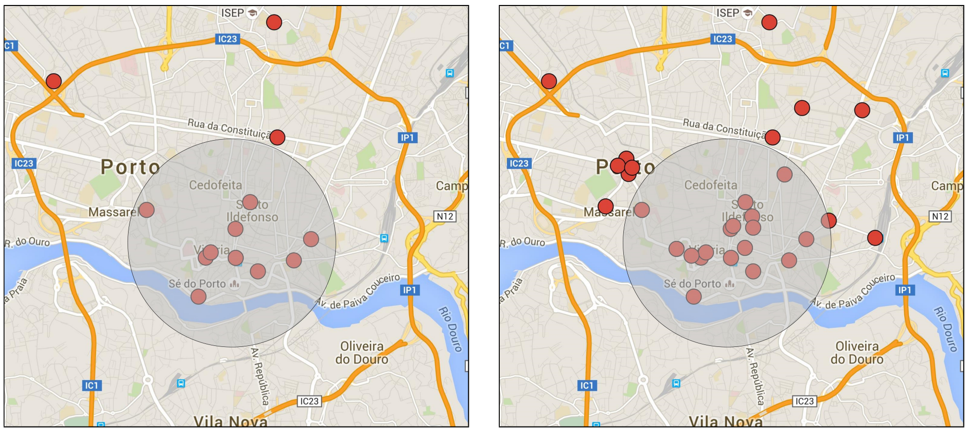


Figure 6.3: Testbed description (February 2015)

As illustrated by the gray circle in Figure 6.4, the first geographical region selected is the Oporto city center. This circle is centered in *Aliados* and has a radius of 1 km. This region is selected due to the high density and mobility of the OBUs and due to the great number of deployed fixed infrastructures (red dots in Figure 6.4). Thus, both emulators considered all the vehicles and RSUs that are within this geographical area during (at least) a sampling time interval.



(a) - October 2014 (b) - February 2015

Figure 6.4: City center scenario

As described in Table 6.7, two periods to disseminate the content through the network were considered: (i) *Rush Hour* and (ii) *non-Rush Hour*. The first one occurs during the morning period (from 6am to 10am) and aims to analyze the behavior of the proposed strategies in a period where the number of nodes is quite high since the number of vehicles traveling in direction to the city center is high. The non-Rush Hour period is framed between 10am and 2pm when the majority of the buses tend to be parked in the parking lots whereby the number of vehicles in the city center is lower than in the previous period. This fact allows an evaluation of the proposed strategies in a scenario characterized by an often intermittent connectivity between the network nodes and a sparse network topology.

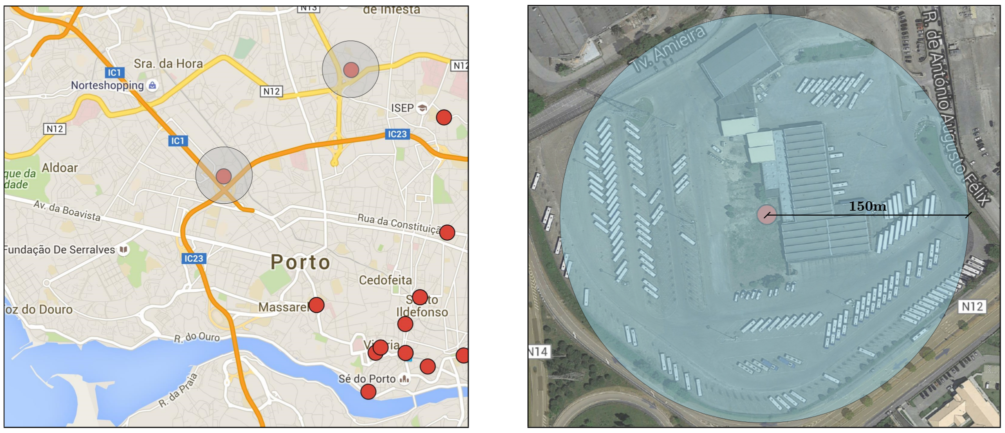
Period		**Region**	**Dataset**	**OBUs**	**RSUs**
Rush Hour	06am-10am	City center	Oct 2014	129	11
			Feb 2015	161	18
non-Rush Hour	10am-02pm		Oct 2014	66	11
			Feb 2015	133	18
Parking	08pm-12pm	Parking lot	Oct 2014	110	1
			Feb 2015	120	1

Table 6.7: Number and type of nodes evaluated for each period and dataset

Parking Lot

The second geographical area that was evaluated is described in Figure 6.5. This scenario aims to evaluate the behavior of the content distribution strategies within a parking lot. In this case, the parking lot of *Via Norte* was selected. Among the two existing ones, this is the biggest parking lot of a public transportation company and can be mapped as a circle with 150 meters radius. This parking lot has an RSU located in its center which is the RSU number 447. In this document, the RSUs located in a parking lot can be also called Collection Station (STA). This geographical area is mainly characterized by a high density of nodes and a very low mobility since most of the time the vehicles are parked. Thus, it is a potential good environment for content distribution due to its high density of nodes but it is expected that the network congestion will be a serious problem.

In this scenario and as described in Table 6.7 only one time period is evaluated. The selected time period is comprised between 8pm and 12pm since this is the period with more traffic in the selected parking lot. There are other interesting periods, however in those periods the nodes do not remain parked for a long period of time, and it is not ideal to perform an evaluation in this kind of network topology.



(a) - Overview at October 2014 (b) - Zoom in the parking lot of RSU 447

Figure 6.5: Parking lot scenario

6.4.2 Laboratory Testbed

The previous scenarios were tested in the developed emulators in order to evaluate the scalability and performance of the proposed strategies. However, the software was developed in order to be implemented in a real vehicular network where the nodes are deployed using the NetRider board previously described. Thus, a testbed was deployed in a laboratory environment in order to test the best strategy that resulted from the evaluation in the emulators. This implementation does not aim to test the scalability of the proposed solution but it is mainly focused on demonstrating that the proposed implementation works in a real environment and not only in the developed emulators.

Figure 6.6 illustrates the deployed scenario in a laboratory environment. This figure represents a draft of the plant of the building number two of the Instituto de Telecomunicações of Aveiro. In this experiment five nodes from two different types of (OBU and RSU) are used. The scenario aims to emulate a real content distribution scenario where the connectivity among the network nodes is intermittent and where there is a node which acts as a mobile data mule (OBU number 107) node to deliver information collected from an access point (RSU number 172) to the remote nodes (OBUs number 109 and 217). The OBU number 132 is fixed and aims to emulate the periods when an OBU is parked in a parking lot where it has direct contact with an RSU. The Table 6.8 resumes the traveling time and distance between different points of the scenario.

From	**To**	**Time [s]**	**Distance [m]**
A	B	15	29
B	C	15	23
C	D	15	25
D	E	15	23
E	B	15	25
B	A	15	29

Table 6.8: Laboratory testbed description

The timeline of the experiment is described in Figure 6.7. The testbed is deployed as a circuit which is delimited by points A to E. The OBU number 107 runs through this circuit three times during the experiment and follows a path that starts in A (where it stays for a period of 30 seconds) and goes through B, C, D, E, B, and ends in the starting point A. This circuit takes an average time of 90 seconds and, along with the stopping period in A, is repeated three times in the same experiment whereby it takes a total time of 6 minutes. Using this deployment it is expected that the mobile OBU downloads the file from the RSU during the period that it is stopped in point A, and disseminates the downloaded content through the sparse nodes of the network (fixed OBUs number 109 and 217). Moreover, the OBU number 132 should download the file faster than the other nodes, since it emulates a parked vehicle with direct contact with an RSU.

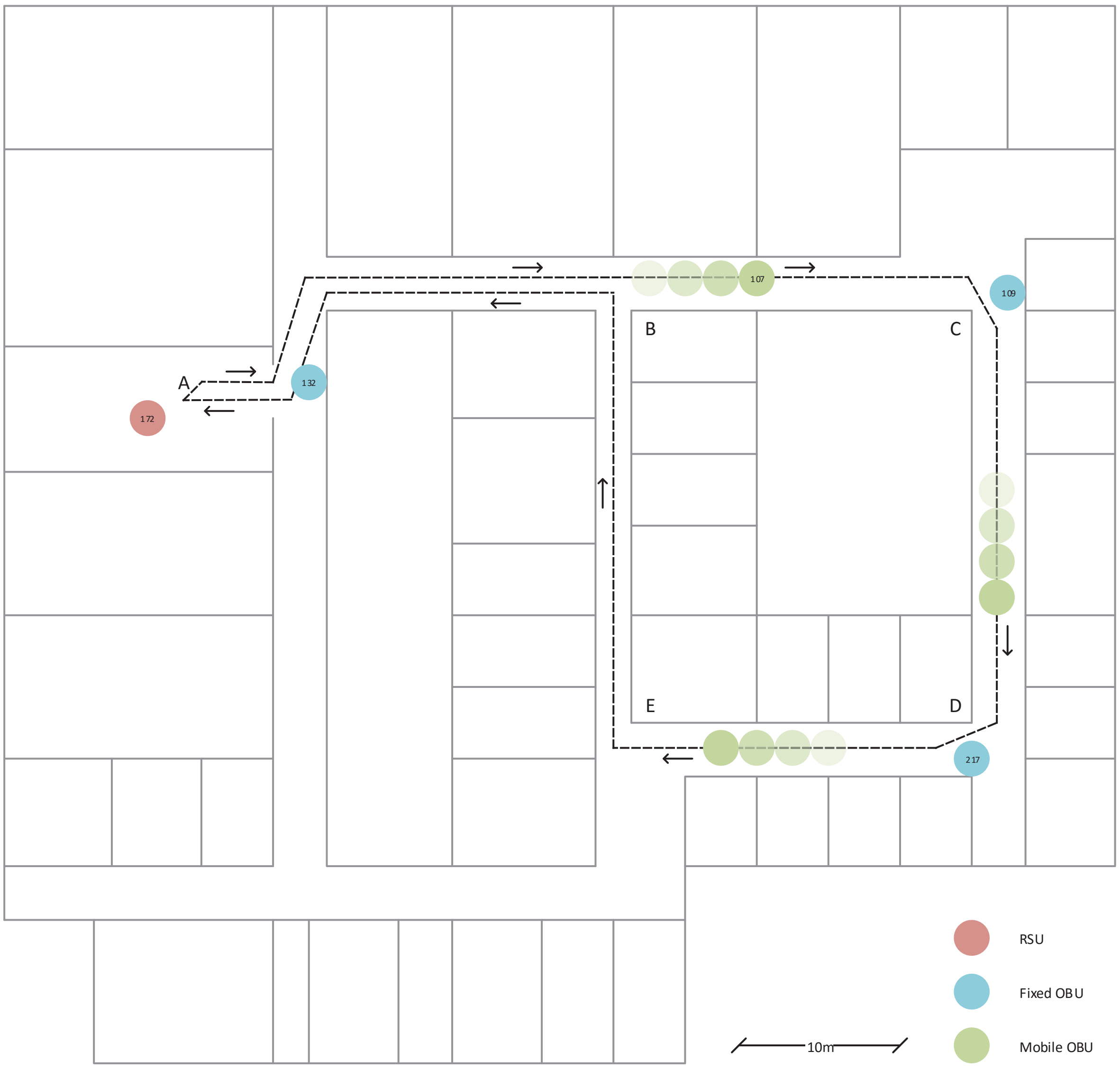


Figure 6.6: Laboratory testbed description

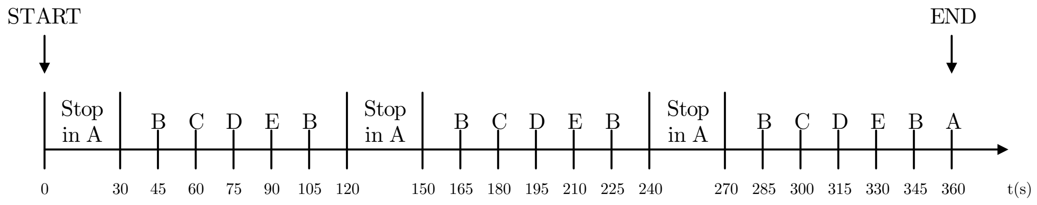


Figure 6.7: Laboratory testbed description - experiment timeline

6.5 Evaluated Metrics

In this section the evaluated metrics are described. These metrics are divided into three major groups: (i) network overview, (ii) content distribution, and (iii) performance. During the evaluation both datasets are analyzed, even though, they are completely independent since they were collected in different periods and have a different number of nodes. Thus, all the metrics must be compared in experiments within the same dataset.

6.5.1 Network Overview Metrics

Before the beginning of the evaluation procedure, it is crucial to better understand what are the main characteristics and specifications associated with the network and testbed under evaluation. Regarding that, several metrics were collected and they are described as follows:

1. Active time is the amount of time that an OBU remains operational, and it is evaluated assuming that each log received (in the collected dataset) represents an increase of T_s seconds (sampling period of the data collection) to this metric.

2. Number of (valid) contacts are the number of times that a certain node can establish a contact with a neighbor. Thus, as an example, when a node at a specific time instant has a vicinity comprising 5 nodes, the total number of contacts registered at this time is 5. However, if among these five, 2 of them have a link connection below 15 dBm and another is an RSU, the number of valid contacts is reduced to 2, since only connections equal or above 15 dBm can establish downstream (from an RSU to an OBU, or among OBUs). This metric can also be represented normalized according to the number of different nodes (OBU or RSU) which collected the information about the number of contacts.

3. The OBU's mobility can be measured using the number of collected contacts between an OBU and other nodes divided according to their type (OBU, RSU, and STA). Thus, if an OBU has a higher number of contacts with an RSU than with a STA, it indicates that it has been traveling more in the city center than parked in the parking lot.

6.5.2 Content Distribution Metrics

During the evaluation procedure the following content distribution metrics were evaluated:

1. Delivery rate is considered the percentage of emulated OBUs that successfully downloaded the content under dissemination, and this metric is evaluated on a hour-by-hour basis.

2. Cumulative percentage of file distributed in the network (all the OBUs) throughout the experiment.

3. E2E delay is the elapsed time to receive the complete file in an OBU since the beginning of the dissemination process. In the plot only the nodes that received all the packets are considered.

4. Progress rate measures the rate of progress of completed download files averaged over all nodes over a specific time interval (default is 20 minutes).

5. Number of listened packets by the network per hour. This metric is evaluated through the cumulative sum of all the packets that the OBUs listened in the medium, and it can be interpreted as a medium congestion metric.

6. Number and size of transmitted advertisement packets by the network (OBUs and RSUs) per hour (only measured in the HelixEmulator).

6.5.3 Performance Metrics

During a heavy simulation procedure it is important to monitor the performance of the simulator machine, since sometimes it could be above its own limits and distort the final results. Thus, during the emulation procedure of the HelixEmulator several performance metrics are collected and evaluated. These are described in the following topics. As a footnote, no performance metric was measured during the MatlabEmulator evaluation, since it works in a sequential way, which guarantees that all tasks are initiated, run, and completed every time.

CPU Usage

CPU usage can be defined as a metric on how much the processor is working and is given in a percentage of CPU time over the total CPU's capability. Also called CPU time, this metric measures the amount of time for which a CPU was used for processing the set of instructions of an OS.

Since the HelixEmulator launches a large set of processes, each one of them with multiple threads, this metric is useful to quantify how the processor is shared between them. Furthermore, it allows to quantify the overall business of the system.

The CPU usage information is collected in a UNIX OS. This information is available in the `/proc/stat` path of the OS. The CPU usage is evaluated by dividing the difference between total CPU time and idle CPU time, with the total CPU time as shown in Equation 6.1. The total and idle times are measured during a specific sampling period (default value is 10 seconds).

$$CPU_{usage} = \frac{TOTAL_{time} - IDLE_{time}}{TOTAL_{time}} \times 100 \tag{6.1}$$

Load

UNIX operating systems have a metric called load which is a measurement of the computational work under performing [165]. In a Linux OS, to evaluate the load, processes waiting for CPU and other resources (e.g. processes waiting to read from or write to the disk) are counted. The instant load does not mean too much since it can vary quite fast. Thus, usually the load average on a certain period of time is evaluated.

In [166] an analogy with load and traffic is made which is very illustrative of the meaning of load. It compared a single-core CPU with a single lane of traffic with a maximum car capacity. Thus, a decent metric would be "how many cars are waiting at a particular time" [166]. If there are cars in lane, the traveling time will increase compared to a situation where there are no cars backed up. So, a possible measure of this behavior is the following classification (i) 0.00, if there is no traffic on the bridge at all, (ii) 1.00, if the lane is exactly at capacity, and (iii) >1.00 if there is backup. So, if the load is 2.00 that means that the exact same number of the current cars within it are waiting to enter in the lane. Thus, Assuming that cars are the processes under execution which use a slice of CPU time or queued up to use the CPU, this analogy describes what the load is.

In the case of multi-processors, the maximum load is given by the number of processor cores available. As an example, if the CPU has 8 processors, the load should always be below 8.0. However, one of the processors is typically dedicated to the operating system, whereby only the remaining processors are allocated to other programs. Thus, it is assumed that the maximum load should be the total number of single-core processors (or cores) minus one.

This measure is given by the UNIX operating systems in the `/proc/loadavg` path, and can refer to a sample period of one, five, and fifteen minutes.

Memory

Another important metric is related to the main memory (RAM) of the OS. When a program/process is executed, the OS assigns a certain memory space, and loads the program binary code into memory. Thus, since multiple processes and threads are running at the same time, it is important to monitor the evolution of memory along the experiment period.

The information is collected through the UNIX command `free`, which gives the total, free, and used memory of the system at this time instant.

6.6 Initial Study of the Network

As previously mentioned, two datasets of information were collected from a real VANET and, in this work, several strategies of content distribution are studied. Thus, regarding a correct understanding of the network, its behavior and characteristics, an initial study is crucial.

This first approach allows a more accurate routing decision and development of protocols to disseminate information through the whole network in a real context. Moreover, this overview is quite useful to adapt the strategies related to network congestion control and optimized delivery, since it provides useful information about the number of contacts of a node and its mobility during a 24h period.

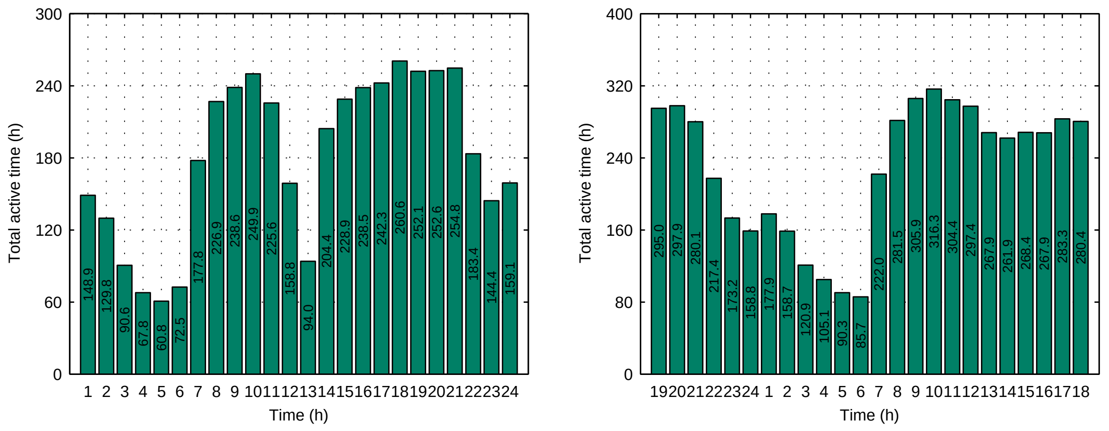
6.6.1 Active Time of OBUs

Figure 6.8 illustrates the total active time of all OBUs during a 24-hour period. As mentioned before two datasets of log information were collected during a 24-hour period, the number of OBUs is refereed in Table 6.7. Thus, in Figure 6.8 each bar represents the total active time of all OBUs in one hour of collection data. The active time is evaluated by multiplying the number of times that a node generates a log information file by the sampling period (T_s). After analyzing this data it is possible to conclude that in both collected datasets, there are periods with a higher number of active OBUs than others.

In October, the periods with a higher number of active nodes occur during the early morning (7am to 12am) and evening (2pm to 8pm). The low level of contacts between 11am and 1pm is justified by the nodes that are parked in the parking lot at the end of the morning service and remain parked until the beginning of the evening.

In February, the active time is more uniform during the day being nearly constant between 7am and 10pm. This profile could be due to the increase in the number of nodes in the dataset. Moreover, a constant decrease of active time is registered between 12pm and 6am since most of the vehicles are parked.

Another important characteristic is the fact that, when a vehicle parks (or stops for any reason), the OBU continues to communicate during a period of approximately one to two hours. Thus, if a vehicle parks at 11pm, it could remain active until 1am. This last fact justifies the still high number of active nodes after midnight.



(a) - October 2014

(b) - February 2015

Figure 6.8: Network overview - Total active time of OBUs

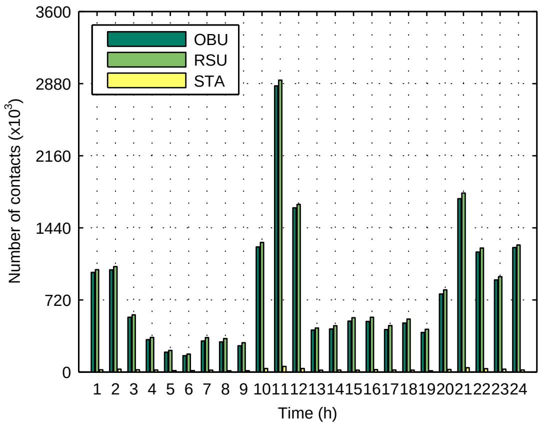
6.6.2 Number of Contacts

Figure 6.9 registers the total number of contacts (with OBUs and RSUs) that a node (OBU) registered per hour. The number of contact are evaluated counting the number of time that a node (OBU) has connection to other nodes (OBUs and RSUs). As an example, if node A has a connection with node B during 10 sampling intervals, the number of contacts will be 10. Thus, the number of contacts is also directly related with the duration of the connection. The results are organized according to the node's type, which means that each bar in an hour period represents a specific node type. The count may contain duplicated values since, if node A registered a contact with node B, both node's counters will be incremented. Another note is the fact that the number of contacts in February will be always higher than in October due to the increasing number of nodes and also to the lower sampling period.

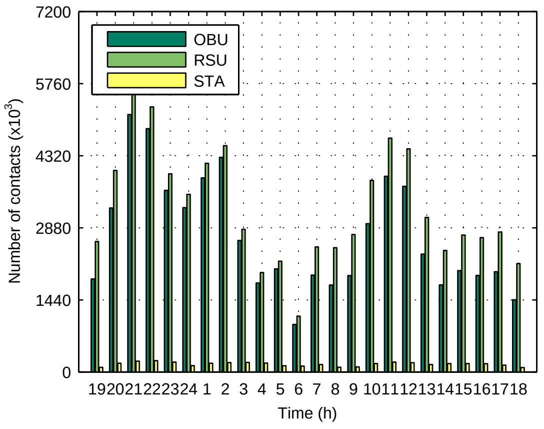
As expected, the periods with a higher number of contacts are nearly similar to the ones where nodes remain most of the time active. Nevertheless, the peaks are more pronounced during the periods around the midday and the beginning of the night. This can result from the simultaneous arrival of multiple vehicles to the same place (parking lots).

However, not all of the contacts are valid to perform a constant dissemination, whereby the previous results must be analyzed excluding the non-valid contacts. In Figure 6.10 all the contacts with a link's RSSI below 15 dBm are not suitable to perform a downstream transmission of data and are excluded, due to the fact that the link quality is decreased and the number of lost packets is significant. This filtering results in a decrease in the number of registered contacts compared to the previous analysis.

On the other hand, the absolute number of contacts is not enough to fully understand what are the types of nodes with a higher number of contacts per hour. Thus, in Figure 6.11, the normalized value of valid contacts per hour is illustrated. The normalization allows for a better understanding of the network behavior since it provides an approximate metric of

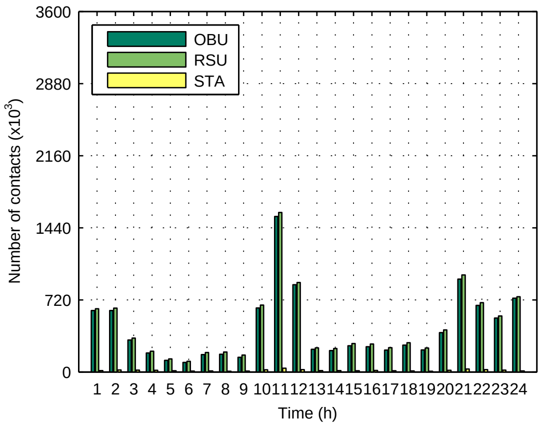


(a) - October 2014

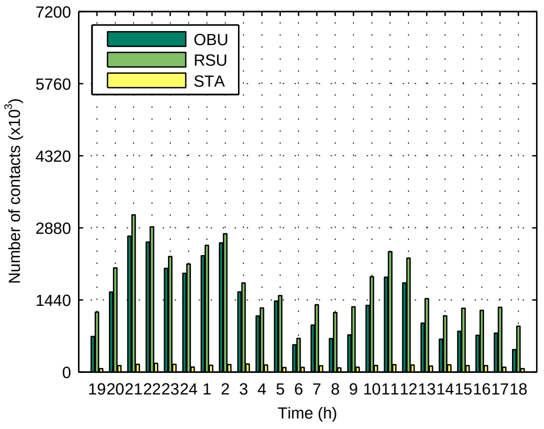


(b) - February 2015

Figure 6.9: Network overview - Number of all type of contacts per hour



(a) - October 2014



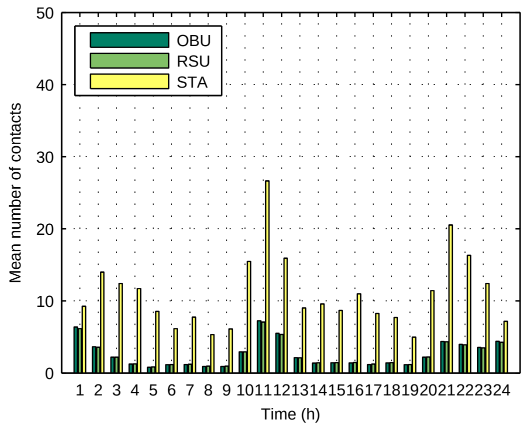
(b) - February 2015

Figure 6.10: Network overview - Number of valid contacts per hour

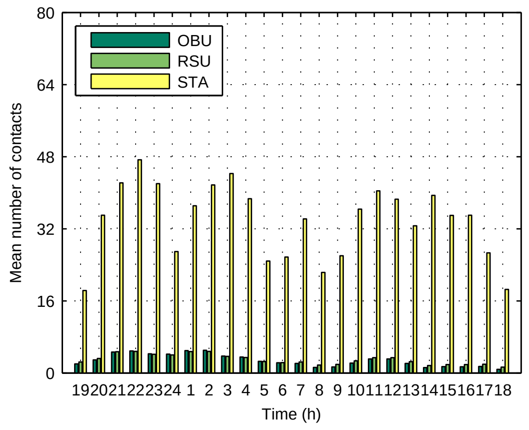
where the nodes stay for a long period of time. So, as illustrated, the STAs have always a higher number of valid contacts because they are located in the parking lots where nodes tend to be stopped for a long period of time. On the other hand, when not parked OBUs tend to be in movement producing a lower number of contacts, since they establish contact for a reduced time period and could be located in *dark-zones* where they do not have any type of neighbors.

6.6.3 Mobility of OBUs

The information about the established contacts can also be useful to better understand the OBUs mobility. When the type of contacts (valid or non-valid) of an OBU with RSUs and STAs are individually analyzed, it is possible to "recreate" their path along the day. Thus, if the majority of contacts are established with a STA, it means that the OBU is mostly parked in a parking lot. On the other hand, if the majority of them are with RSUs, the probability



(a) - October 2014



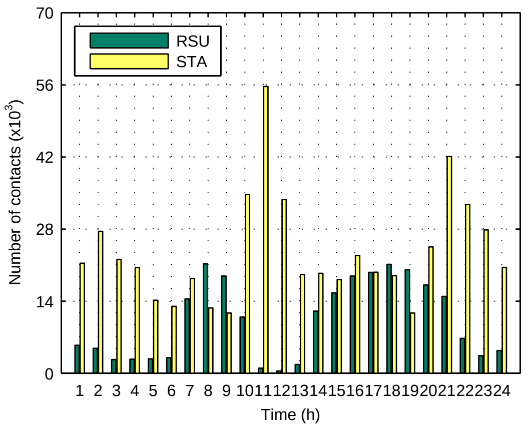
(b) - February 2015

Figure 6.11: Network overview - Normalized number of valid contacts per hour

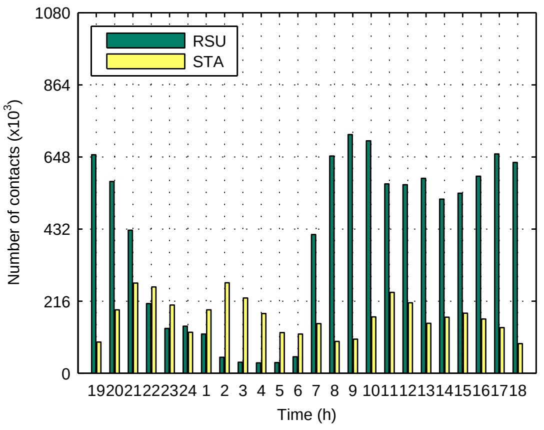
that the OBUs have been travelling in the city is quite high.

Figure 6.12 illustrates the previously described metric. Due to the increase of the number of RSUs between October and February, the presented results are quite different, although the conclusions about the mobility of the OBUs are nearly the same. Thus, in both datasets two scenarios can be clearly identified: OBUs located mostly (i) in the city, and (ii) in the parking lots.

In both datasets the OBUs start to have more contacts with STAs approximately after 9pm and remain parked until 6pm. After 6am the OBUs start traveling through the city and, at midday, some of them returned to the parking lots where they remained for a period of one or two hours and then returned to the city. This behavior is more clearly observed in October due to the lower number of RSUs located in the city when compared to the latest period of February. This higher number increases the number of contacts between OBUs and RSUs during the day, providing a better coverage to the vehicular network.



(a) - October 2014



(b) - February 2015

Figure 6.12: Network overview - Mobility of OBUs

6.7 MatlabEmulator Evaluation

In the initial phase of this work the MatlabEmulator was developed. The main idea behind its development was to create an easy platform to evaluate a variety of strategies and approaches to disseminate content through a vehicular network. Thus, in this section a set of results and conclusions resulting from that study are presented and discussed.

The approach to evaluate the proposed strategies is divided in two. The first one is related to "Stateless Choose Information" and aims to study the individual behavior of the strategies described in section 4.3. The second one is focused on solving problems related with a high congestion of the medium without jeopardizing the delivery of information (see section 4.4).

6.7.1 Strategies to Stateless Choose Information

All four proposed strategies to stateless choose information are evaluated. The standard experiment is characterized by the dissemination of a 100 MB file divided in 3008 packets of 32 KB each. The three previously described scenarios are evaluated: (i) Rush Hour, (ii) non-Rush Hour, and (iii) Parking.

In this emulator the immediate sending and reception of the advertisement packets is assumed whereby all the nodes have all the necessary information to perform a routing decision. Thus, in the LRBF and LRGF strategies the advertisement process is implicit, and it is assumed that the nodes know exactly what their neighbors have stored, being this an ideal scenario. In order to approximate the emulator's behavior to the real behavior of this type of service in the real network the bandwidth is limited to 1 Mbps.

Moreover, in the LRGF strategy only the second situation (see section 4.3) that considers the possibility of forwarding coded packets by other nodes is evaluated, since it presented a better performance compared to the first one. In this strategy, a block is composed by 8 packets and, when coded, results in a generation of 12 coded packets since these are the most common values in the literature.

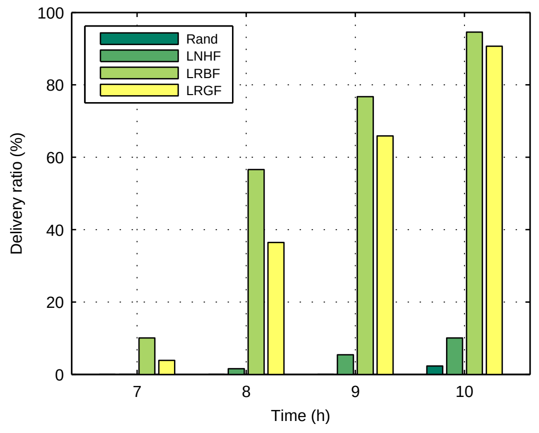
6.7.1.1 Rush Hour Period

Both Figure 6.13 and Figure 6.14 represent metrics associated with the percentage of the delivery along the rush hour period. The first one only counts the number of nodes which have completely received the file, and the second one is related to the percentage of file spread through the network (even if the file is not completed).

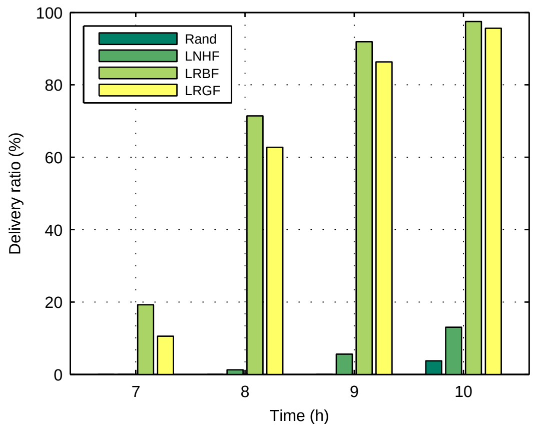
Analyzing the results it is clear that LRBF and LRGF have the best behavior in terms of delivery. On the other hand, the Random strategy is quite inefficient since it can not completely deliver the file to almost any nodes. The LNHF strategy appears in the middle achieving a higher delivery of complete files than the Random strategy.

All the strategies converge to a high percentage of files distributed in the network. However, the Random and LNHF strategies take more time than the other two to achieve the same percentage. Thus, it can be concluded that the intelligence introduced in the LRBF and LRGF strategies by the advertisement packets can lead to a better performance in terms of delivery of completed files.

The next two metrics (Figure 6.15 and Figure 6.16) are related to the delivery time. The first one represents the time that each node tokes to download the complete file, whereby only information about that node is displayed. The second metric is the progress rate and aims to give a perspective about how quick is the delivery of the file in each one of those nodes.

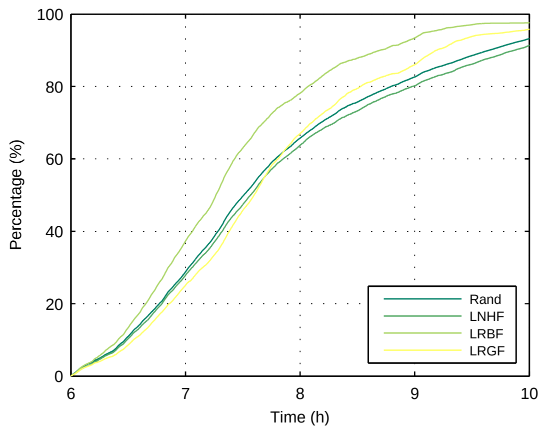


(a) - October 2014

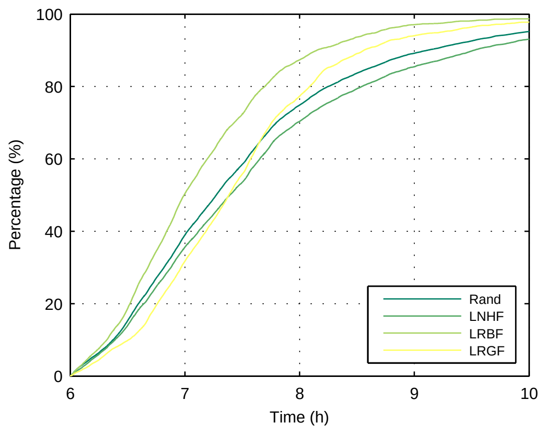


(b) - February 2015

Figure 6.13: MatlabEmulator evaluation comparison in rush hour period - Percentage of nodes with complete file per hour - Delivery rate



(a) - October 2014



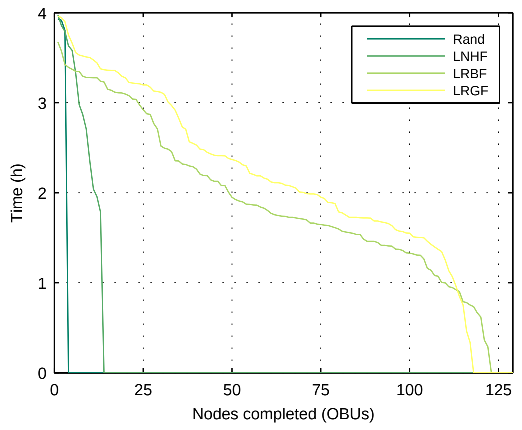
(b) - February 2015

Figure 6.14: MatlabEmulator evaluation comparison in rush hour period - Percentage of file distributed in network throughout the experiment

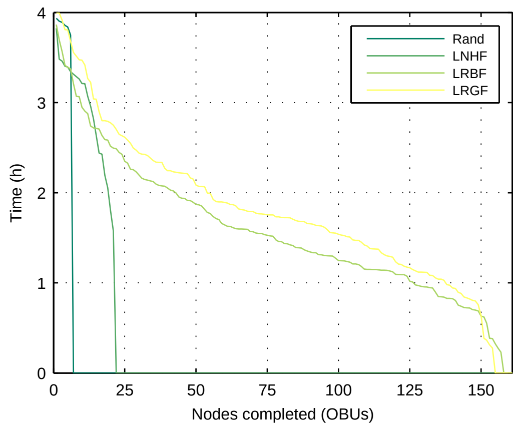
Thus, analysing the results and based on the statistical analysis of Table 6.9, it can be concluded that the LRBF and LRGF strategies have the best performance since, in mean, the download process is faster than in the others. On the other hand, the random and LNHF strategies have the worst behavior. Moreover, as the progress rate shows, in the two better strategies (LRBF and LRGF) the majority of the downloads are concluded in the first two hours. Contrarily, most of the complete downloads of random and LNHF strategies occurs during the last half of the experiment.

Finally a metric regarding the evaluation of the medium congestion is analyzed and illustrated in Figure 6.17. Thus, every time that a node (only OBUs) listens to a packet a counter is incremented, whereby if this value is higher in a specific strategy than in another, it means that the medium is more congested.

The results of the two datasets are quite different. This mismatch can be justified by a



(a) - October 2014

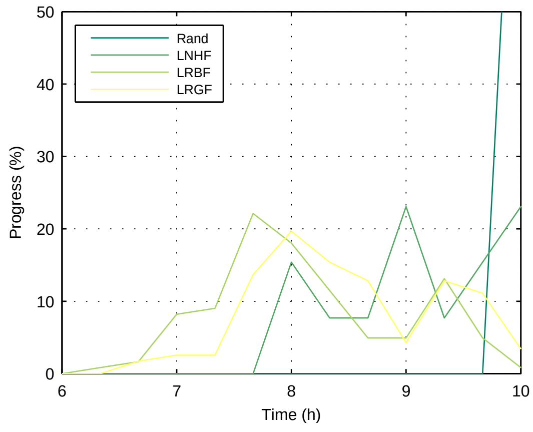


(b) - February 2015

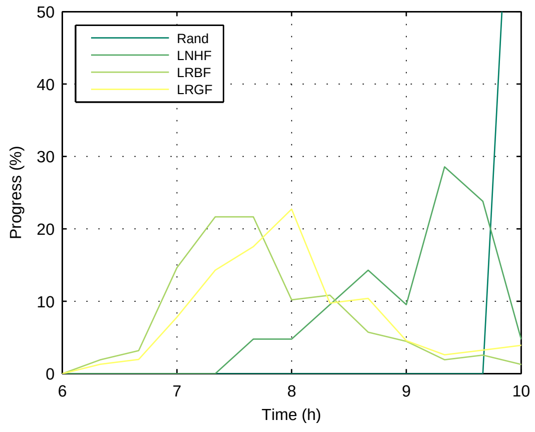
Figure 6.15: MatlabEmulator evaluation comparison in rush hour period - Time to receive the complete file per node - E2E delay

Strategy	October 2014		February 2015	
	Avg [h]	C.I. (95%) [h]	Avg [h]	C.I. (95%) [h]
Rand	3.88	±0.17	3.86	±0.07
LNHF	2.99	±0.47	2.91	±0.28
LRBF	1.98	±0.14	1.61	±0.12
LRGF	2.31	±0.15	1.86	±0.13

Table 6.9: MatlabEmulator evaluation comparison in rush hour period - E2E delay statistics



(a) - October 2014



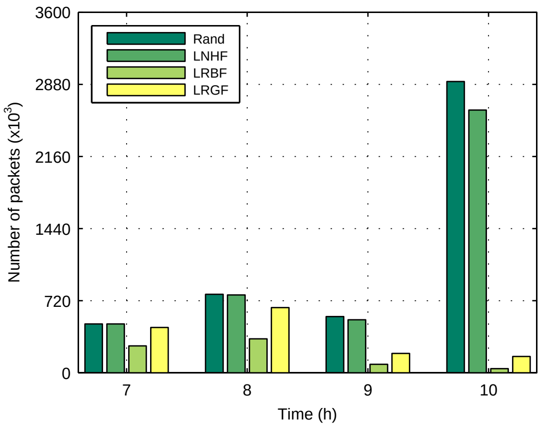
(b) - February 2015

Figure 6.16: MatlabEmulator evaluation comparison in rush hour period - Progress rate

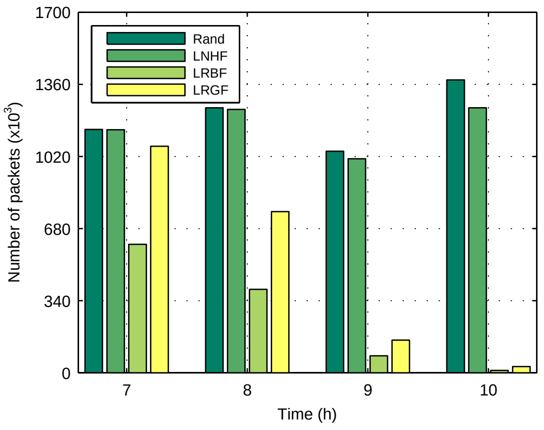
---

different network behavior in terms of number of contacts. Remembering the initially study of the network, in October there is a high increase in the number of contacts between 9am and 10pm which justified the sudden increase of listened packets in this period. Since in February the network profile in terms of number of contacts is more stable and flat, the number of listened packets in the Random and LNHF are approximately equal and constant throughout the experiment.

This increasing is only verified in the Random and LNHF strategies due to their lack of intelligence, which means that, if they have more valid contacts (opportunities to transmit packets), they send more packets. On the other hand, there is a common behavior among the two datasets which is the decreasing of the listened packets in the last periods of the experiment when the LRBF and LRGF are used. This decreasing is justified by the fact that the transmission of packets decrease when the majority of a node's vicinity has downloaded the file. In this situation the node does not send any packet once no node needs any more packets. Furthermore, since the delivery ratio is higher and the end-to-end delay is lower in the LRBF strategy, it has a lower number of listened packets because of the last giving reason.



(a) - October 2014



(b) - February 2015

Figure 6.17: MatlabEmulator evaluation comparison in rush hour period - Number of listened packets by the network (only OBUs) throughout the experiment

6.7.1.2 non-Rush Hour Period

It is now important to evaluate a scenario with a very different profile from the previous, where there is a lower number of vehicles and consequently a decrease in the number of contacts. This behavior is more pronounced in October than in February due to the reinforcement of the network infrastructure which leads to an increase in connectivity among nodes. The following results aim to evaluate the performance of the proposed strategies in this period. The analysis is structured as the previous one, being focused on the delivery ratio, delivery delay, and network congestion.

Figure 6.18 and Figure 6.19 show that the delivery rate in this period when in October is very reduced. This results from a lack of connectivity during the path from the city center to the parking lots where the vehicles remained parked during the midday. The abrupt increase in the delivery rate in the last hour of the experiment is due to the arrival of multiple vehicles into the city center after a period of being parked. On the other hand, analyzing the delivery rate in the dataset of February is clearly understanding the effect of the reinforcement of the infrastructures. Thus, during the travel to the parking lots, the vehicles pass through a higher number of RSUs enhancing the delivery. In this case the behavior is similar to the one previously discussed to the rush hour period.

The tendency among the strategies remains according to the one observed in the rush hour period. Thus, the LRBF presents the higher delivery rate, followed by the LRGF, and with a very low delivery rate, the LNHF and Random strategies.

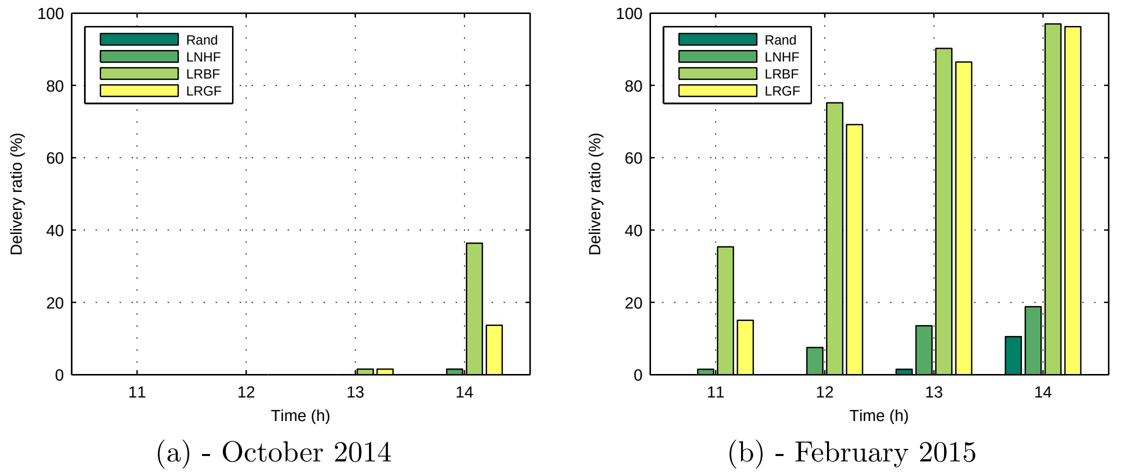


Figure 6.18: MatlabEmulator evaluation comparison in non rush hour period - Percentage of nodes with complete file per hour - Delivery rate

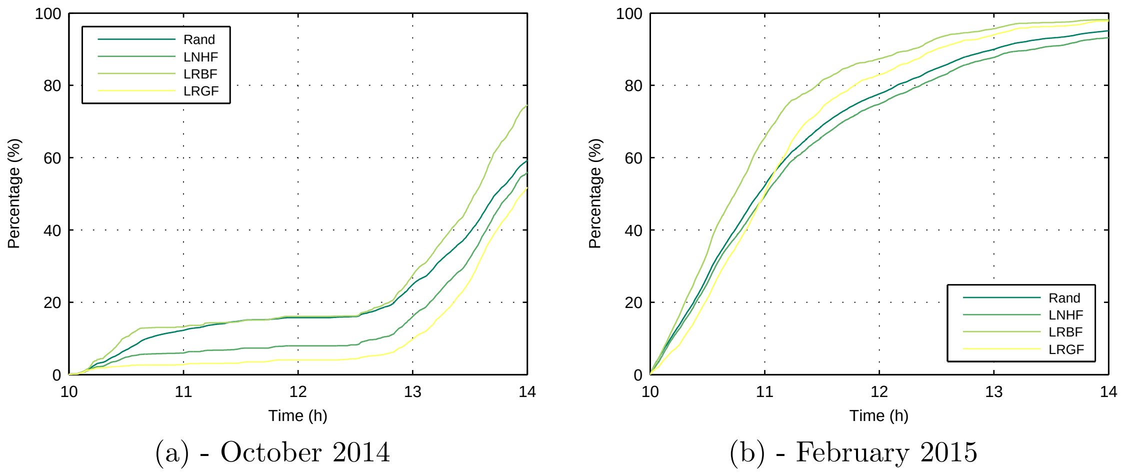
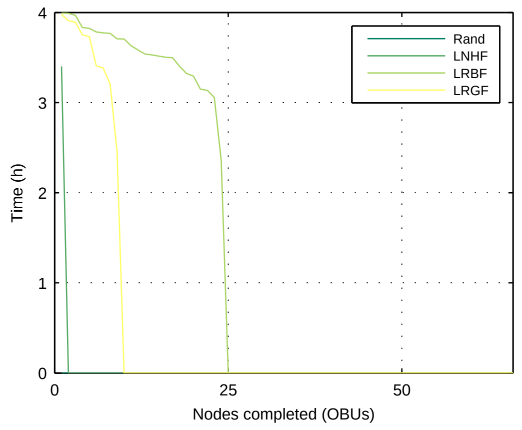


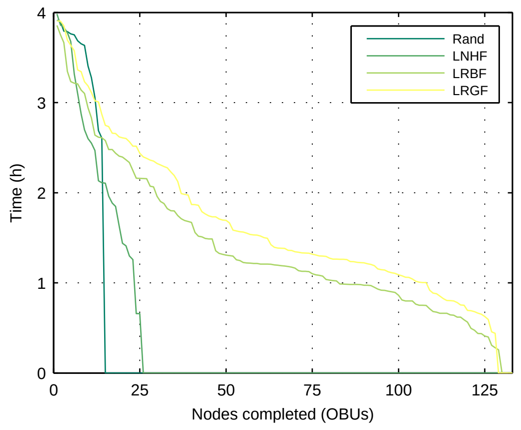
Figure 6.19: MatlabEmulator evaluation comparison in non rush hour period - Percentage of file distributed in network throughout the experiment

In terms of the delay in the delivery in the experiment of October, the majority of the files are downloaded in the final hour due to the reasons stated above. On the other hand, in the February dataset the behavior is similar to the one registered during the rush hour period of February. Moreover, as Table 6.10 shows, the strategies with a better behavior are the LRBF and LRGF strategies, followed by the LNHF and Random strategies. Figure 6.20 and Figure 6.21 prove the previous statement.

Analyzing the metrics related to the network congestion presented in Figure 6.22, it is also



(a) - October 2014

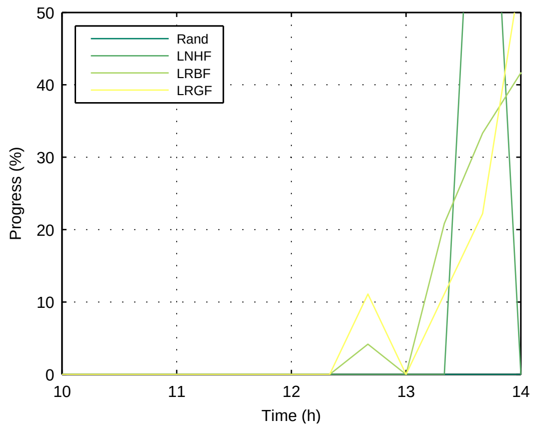


(b) - February 2015

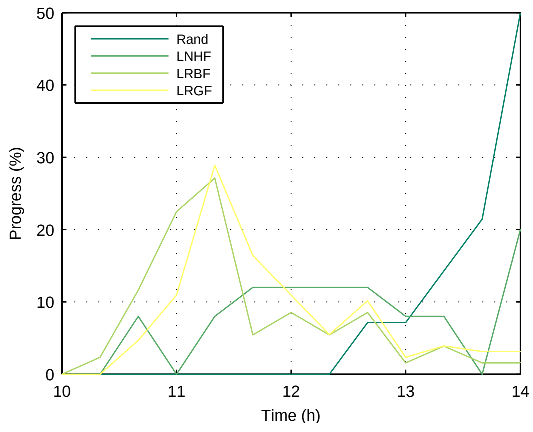
Figure 6.20: MatlabEmulator evaluation comparison in non rush hour period - Time to receive the complete file per node - E2E delay

Strategy	October 2014		February 2015	
	Avg [h]	C.I. (95%) [h]	Avg [h]	C.I. (95%) [h]
Rand	-	-	3.49	±0.25
LNHF	3.40	-	2.37	±0.41
LRBF	3.54	±0.15	1.43	±0.14
LRGF	3.54	±0.37	1.69	±0.14

Table 6.10: MatlabEmulator evaluation comparison in non rush hour period - E2E delay statistics



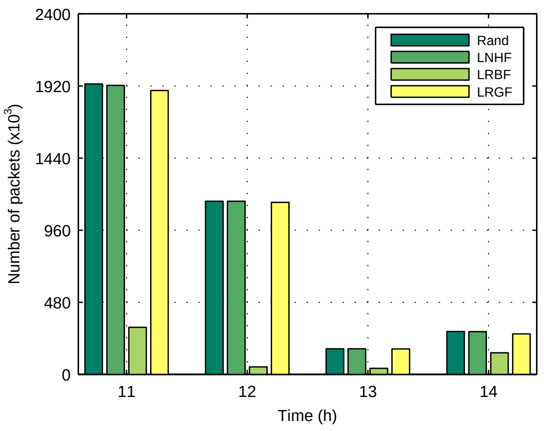
(a) - October 2014



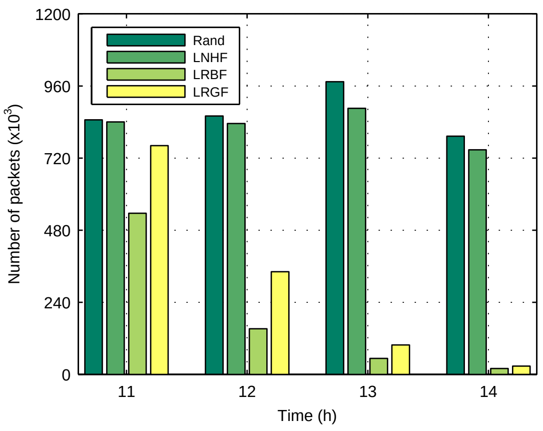
(b) - February 2015

Figure 6.21: MatlabEmulator evaluation comparison in non rush hour period - Progress rate

possible to detect a different behavior from October to February. Due to the high number of contacts during the 10am to 12am period in October, the number of listened packets is higher in this period, decreasing at the end of the non rush hour period. In February, the behavior is more constant since the number of contacts are also flat. Nevertheless, in both periods a gradual decrease of listened packets in the LRBF and LRGF strategies was registered, due to the intelligence introduced which allows a more accurate decision of transmissions. As the number of contacts remains stable, the number of listened packets in Random and LNHF also follows this behavior, since they are closely dependent on the number of contacts sending packets every time that a node can contact another one.



(a) - October 2014



(b) - February 2015

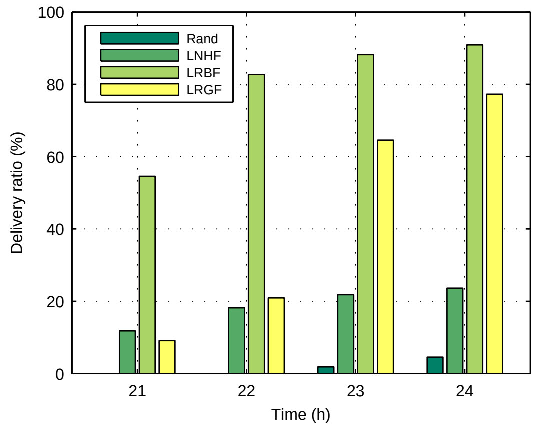
Figure 6.22: MatlabEmulator evaluation comparison in non rush hour period - Number of listened packets by the network (only OBUs) throughout the experiment

6.7.1.3 Parking Period

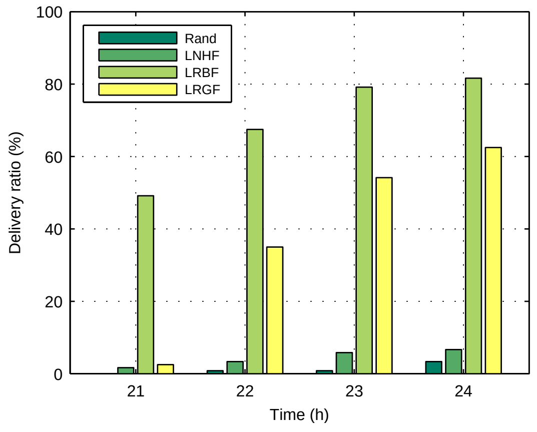
This period has several specificities quite different from the previous two. This is the time period where the majority of the vehicles (buses) arrive into the parking lot to be parked until the morning of the next day. Thus, as seen in the initial network study, this period presents a higher number of contacts since the vehicles are parked in the same geographical region and remain active for (at least) one hour after the engine stops. As in the previous periods the analysis of the parking period is divided in three aspects: delivery ratio, delay in the delivery, and network congestion.

Regarding the delivery ratio, this period has a slightly different profile comparing to the other two. As Figure 6.23 and Figure 6.24 illustrate, this difference is more prominent in the first dataset than in the February period. In October, in the first two hours of the experiment the LNHF strategy has a delivery ratio close to the LRGF strategy, although after that the delivery ratio of LRGF increases a lot and the number of completed files in the LNHF strategy has a gentle increase. Once again, this different behavior is justified by the lack of intelligence of the LNHF strategy which reveals its inability to successful completely delivery a high number of files even when the overall percentage of the file distributed in the network is quite high. The strategy with a better delivery ratio is the LRBF, followed by the LRGF and LNHF.

The major difference regarding the previous periods is the fact that the difference between the LRBF and the LRGF increased significantly. This change of behavior is mainly justified by two things. The first one is related to the latency of the strategies. The LRGF needs to collect `blocksize` coded packets in order to decode the associated block, whereby the process is slower. The second one is due to the high concentration of nodes (this period has a higher number of contacts) which enhances the performance of the LRBF since the ideal advertisement procedure is assumed. Thus, a node knows exactly which is the storage content of its vicinity, and could send (in broadcast) the most lacking packets.

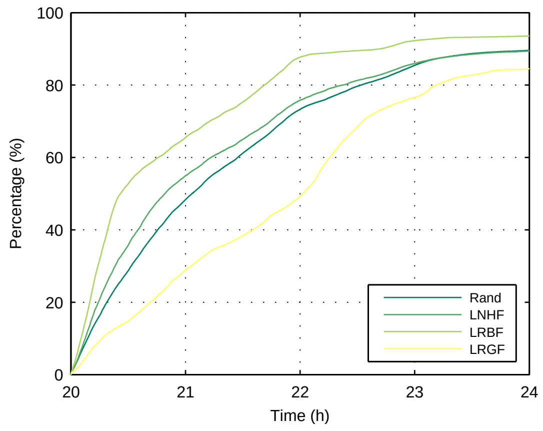


(a) - October 2014

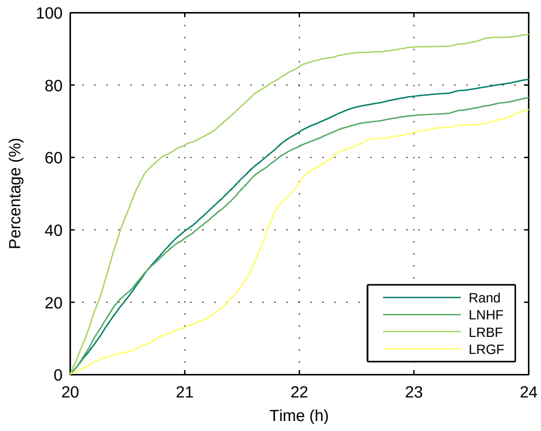


(b) - February 2015

Figure 6.23: MatlabEmulator evaluation comparison in parking period - Percentage of nodes with complete file per hour - Delivery rate



(a) - October 2014



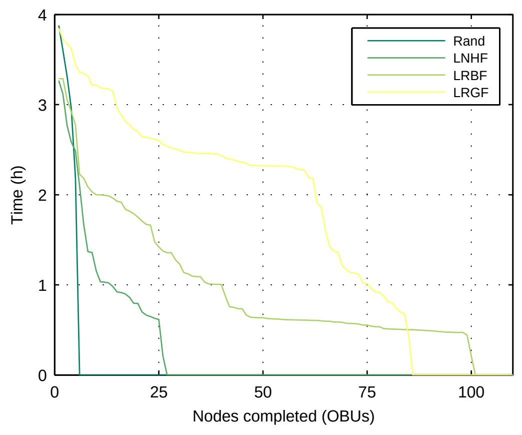
(b) - February 2015

Figure 6.24: MatlabEmulator evaluation comparison in parking period - Percentage of file distributed in network throughout the experiment

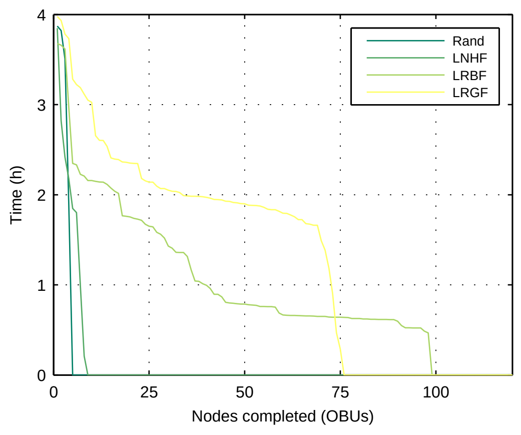
The delay in the delivery is also different from the previous periods as shown in Figure 6.25 and Figure 6.26. Despite the lower delivery ratio, the LNHF strategy presents a lower delay in the delivery when compared to the LRGF, since it delivers mostly in the first period of the experiment (see Table 6.11). Once again, the LRBF strategy has the lowest delivery end-to-end delay of the four evaluated approaches.

Compared to the other evaluated periods, in general, the file is delivered with a a lower end-to-end delay due to the high concentration of nodes within the same geographical area.

Moreover, the frequent contact with the RSU deployed in the parking lot helps in the decrease of this metric.



(a) - October 2014

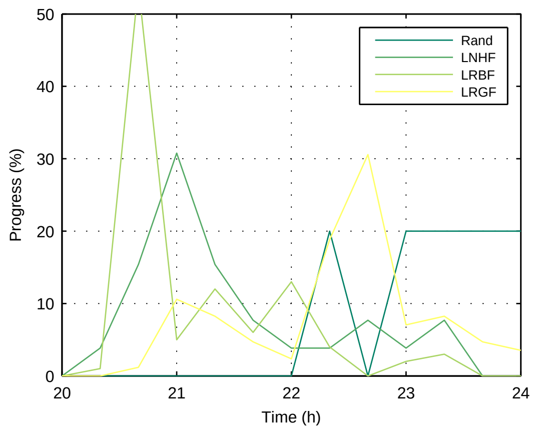


(b) - February 2015

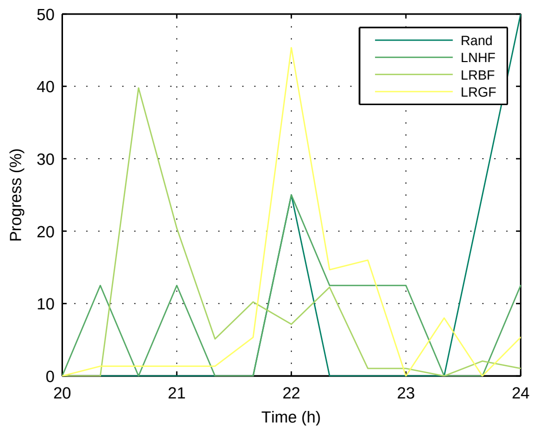
Figure 6.25: MatlabEmulator evaluation comparison in parking period - Time to receive the complete file per node - E2E delay

Strategy	**October 2014**		**February 2015**	
	Avg [h]	**C.I. (95%) [h]**	**Avg [h]**	**C.I. (95%) [h]**
Rand	3.19	±0.81	3.27	±1.50
LNHF	1.33	±0.34	2.02	±0.93
LRBF	1.04	±0.14	1.17	±0.15
LRGF	2.25	±0.18	2.13	±0.15

Table 6.11: MatlabEmulator evaluation comparison in parking period - E2E delay statistics



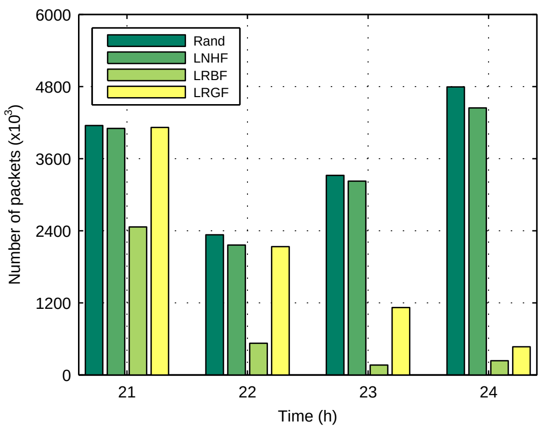
(a) - October 2014



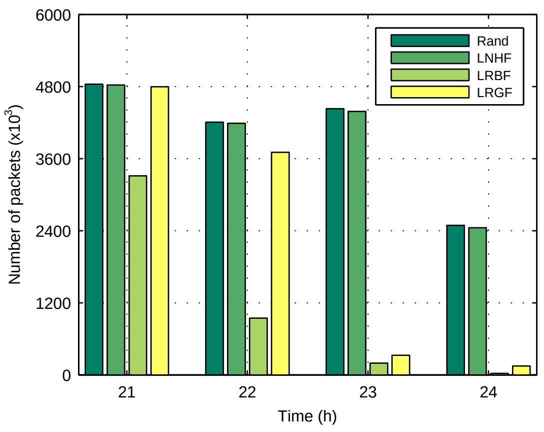
(b) - February 2015

Figure 6.26: MatlabEmulator evaluation comparison in parking period - Progress rate

Figure 6.27 describes the evolution of the number of listened packets by the network throughout the experiment. The same conclusions can be taken from both the October and February datasets. In the periods with a higher number of contacts, the number of listened packets is also high in the Random and LNHF strategies. On the other hand, in both periods a gradual decrease of listened packets in the LRBF and LRGF strategies was registered due to their intelligence which allows for a more accurate routing decision. Moreover, the LRBF strategy always presents a lower number of listened packets compared to the LRGF strategy, which means a lower network congestion (better performance).



(a) - October 2014



(b) - February 2015

Figure 6.27: MatlabEmulator evaluation comparison in parking period - Number of listened packets by the network (only OBUs) throughout the experiment

6.7.1.4 Analysing the Impact of Content Distribution Parameters

After a detailed analysis of the content distribution strategies configured as default, it is important to analyze the impact of some parameters in the dissemination of a content. Thus, in this subsection two parameters are analyzed: (i) the size of the file to be disseminated, and (ii) block and generation size. Only the LRBF and LRGF strategies are evaluated, since they have the best overall behavior in the previous analysis.

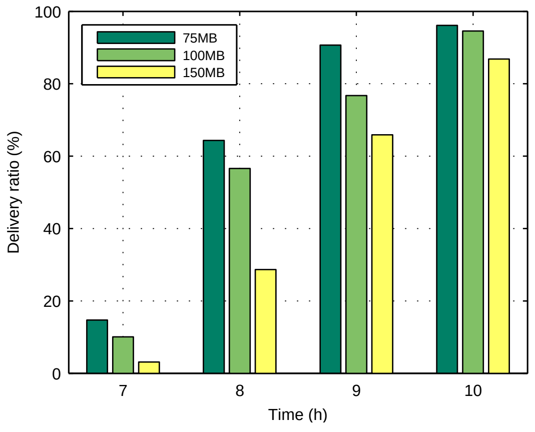
Impact of the File Size on the Delivery Rate and Delay

The most important factor in a dissemination process is the content under dissemination. For example, it is crucial to know if a content is to be disseminated in real-time, which size it has, what is its type, etc. Thus, the impact of the content size in the dissemination process is evaluated. Three file sizes are used: (i) 75 MB, (ii), 100 MB, and (iii) 150 MB.

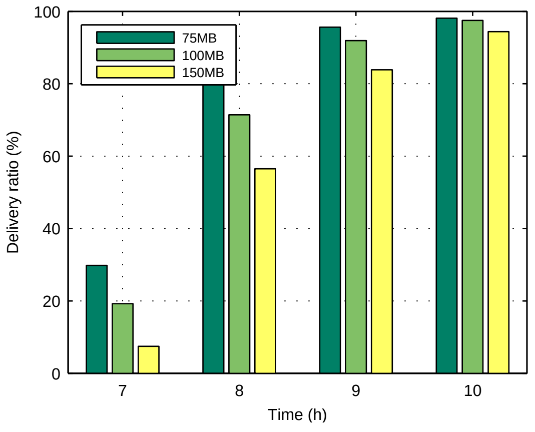
The analysis focuses on the delivery rate and delay impact and all the three dissemination periods are evaluated. All the periods present the same behavior when confronted with different file sizes whereby all of them are analyzed simultaneously.

Regarding the delivery rate (see Figure 6.28, Figure 6.29, and Figure 6.30), it is clear that there is an unequivocal relation with the content size. The bigger the content size, the lower the delivery rate. The highest delivery rate was registered in the dissemination of the 75 MB file, and the lowest one in the experiment of the 150 MB. On the other hand, all the experiments converge to near values of delivery rate. Thus, it can be concluded that the

LRBF strategy successfully disseminated a 150 MB through the majority of the network in a four hour period. The only exception is the non-rush hour period in the October dataset.

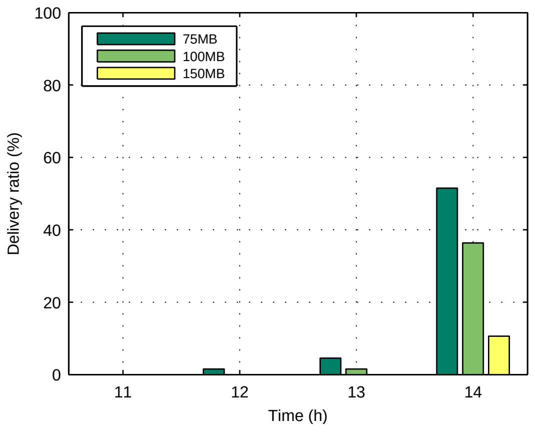


(a) - October 2014

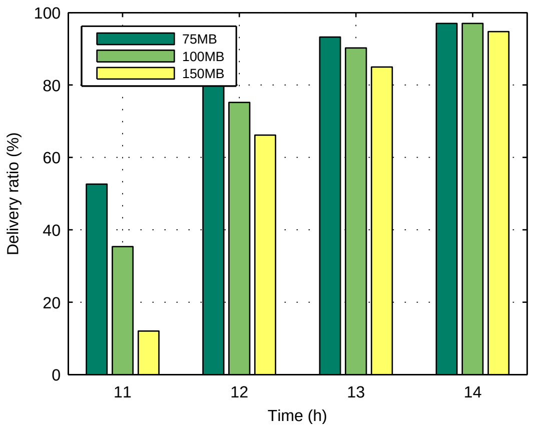


(b) - February 2015

Figure 6.28: MatlabEmulator evaluation impact of the file size during the rush hour period - Percentage of nodes with complete file per hour - Delivery rate



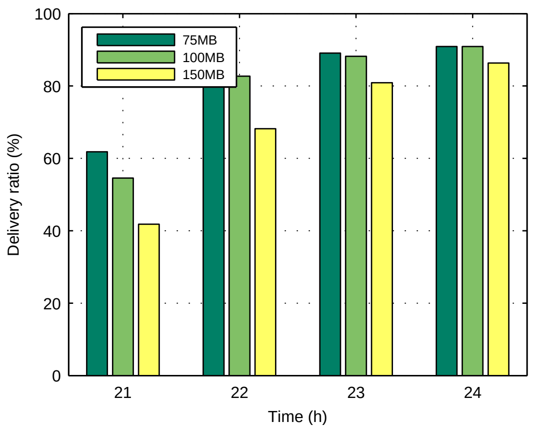
(a) - October 2014



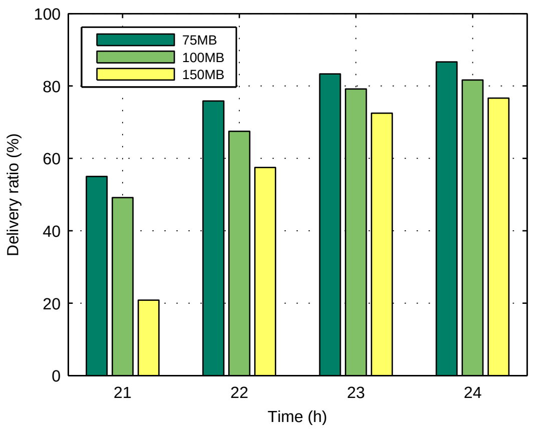
(b) - February 2015

Figure 6.29: MatlabEmulator evaluation impact of the file size during the non-rush hour period - Percentage of nodes with complete file per hour - Delivery rate

In terms of the delivery delay, the Figure 6.31, Figure 6.32, and Figure 6.33 show that the delivery is slower when the content size increases. A more detailed analysis is described in Table 6.12, Table 6.13, and Table 6.14. This analysis corroborates the previous statement, and it is clear that the file size increases the delivery delay.

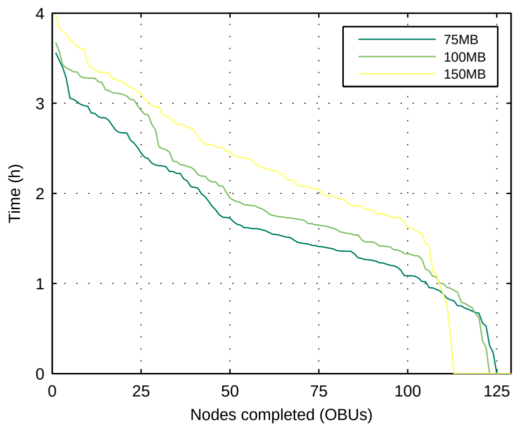


(a) - October 2014

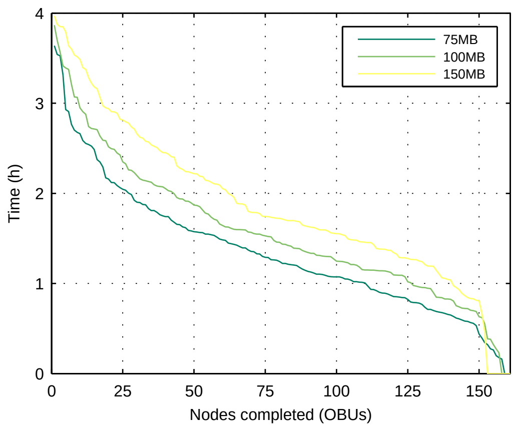


(b) - February 2015

Figure 6.30: MatlabEmulator evaluation impact of the file size during the parking period - Percentage of nodes with complete file per hour - Delivery rate



(a) - October 2014

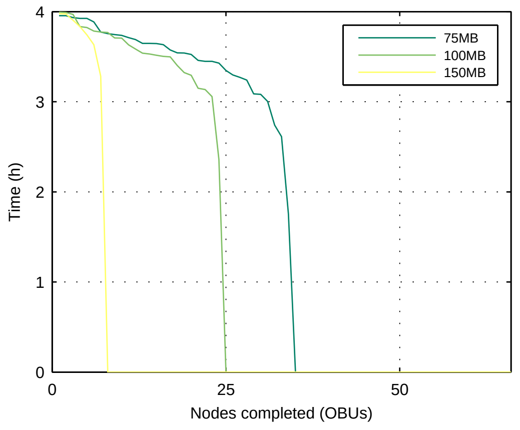


(b) - February 2015

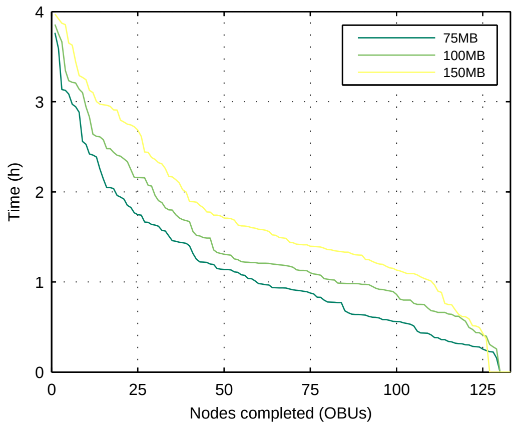
Figure 6.31: MatlabEmulator evaluation impact of the file size during the rush hour period - Time to receive the complete file per node - E2E delay

File Size	**October 2014**		**February 2015**	
	Avg [h]	**C.I. (95%) [h]**	**Avg [h]**	**C.I. (95%) [h]**
75 MB	1.73	±0.13	1.37	±0.11
100 MB	1.98	±0.14	1.61	±0.12
150 MB	2.41	±0.14	1.96	±0.13

Table 6.12: MatlabEmulator evaluation impact of the file size during the rush hour period - E2E delay statistics

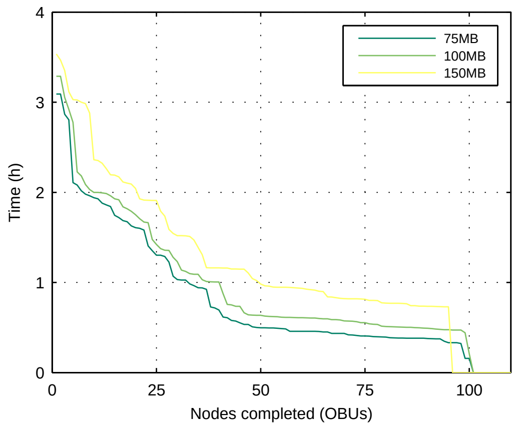


(a) - October 2014

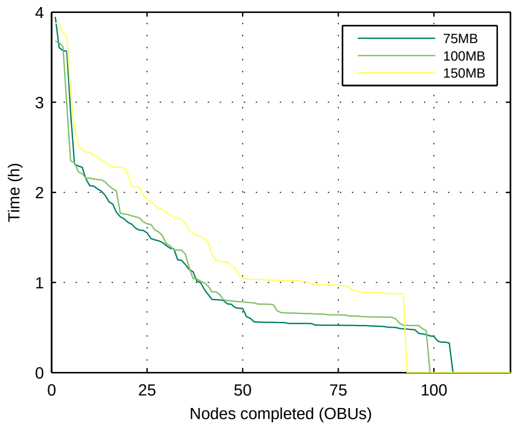


(b) - February 2015

Figure 6.32: MatlabEmulator evaluation impact of the file size during the non-rush hour period - Time to receive the complete file per node - E2E delay



(a) - October 2014



(b) - February 2015

Figure 6.33: MatlabEmulator evaluation impact of the file size during the parking period - Time to receive the complete file per node - E2E delay

File Size	**October 2014**		**February 2015**	
	Avg [h]	**C.I. (95%) [h]**	**Avg [h]**	**C.I. (95%) [h]**
75 MB	3.47	±0.16	1.15	±0.14
100 MB	3.54	±0.15	1.43	±0.14
150 MB	3.76	±0.23	1.77	±0.15

Table 6.13: MatlabEmulator evaluation impact of the file size during the non rush hour period - E2E delay statistics

File Size	**October 2014**		**February 2015**	
	Avg [h]	**C.I. (95%) [h]**	**Avg [h]**	**C.I. (95%) [h]**
75 MB	0.89	±0.14	1.05	±0.15
100 MB	1.04	±0.14	1.17	±0.15
150 MB	1.39	±0.15	1.55	±0.16

Table 6.14: MatlabEmulator evaluation impact of the file size during the parking period - E2E delay statistics

Impact of Block and Generation Size on the Delivery Rate and Delay

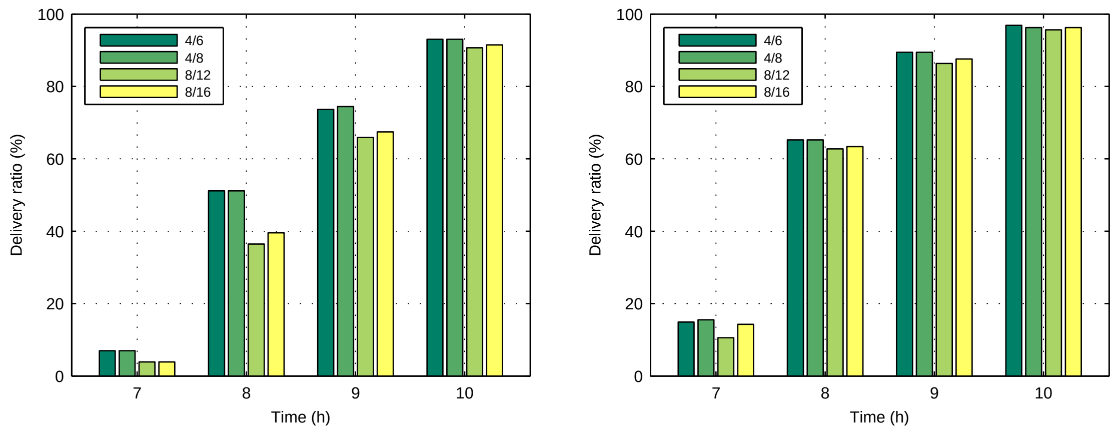
Another important factor when the LRGF is used, is the block and generation size. Thus, the impact of the block and generation size in the dissemination process is evaluated. Four combinations of values (blocksize/gensize) are used: (i) 4/6, (ii) 4/8, (iii) 8/12, and (iv) 8/16. These values are based on the most common approaches used in the literature which selects a 4 or 8 block size with a redundancy of 50% or 100%. The content under dissemination is a 100 MB file.

The analysis focuses on the delivery rate and delay impact and all the three dissemination periods are evaluated. All the periods present the same behavior when confronted with the different combinations of block and generation size, whereby all of them are analyzed simultaneously.

Regarding the delivery rate (see Figure 6.34, Figure 6.35, and Figure 6.36), there is an unequivocal relation with the block size. When the block size is 4, the delivery rate tends to be higher than when the size is 8. A node can download faster the content if the number of coded packets that it must collect in order to decode an entire block is smaller.

On the other hand, the introduced redundancy is defined by the generation size, since the number of additional coded packets is set according to this value. Thus, as the results show, the higher the generation size, the higher the delivery rate, since there are more coded bundles in the network, increasing the dissemination redundancy. However, the impact of the generation size is smaller compared to the block size.

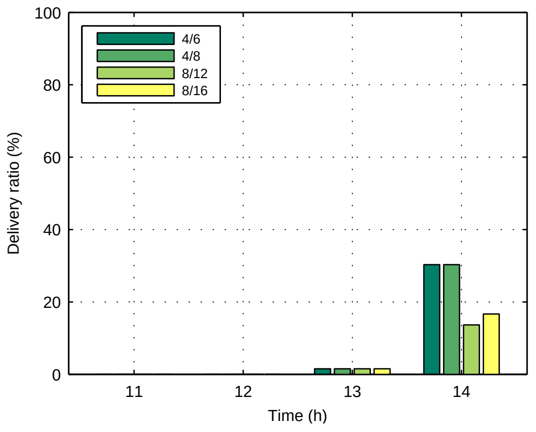
All the experiments converge to near values of delivery rate. Thus, it can be concluded that the LRGF strategy successfully disseminated a 100 MB file in a four hour period independently of the (blocksize,gensize) pair of values.



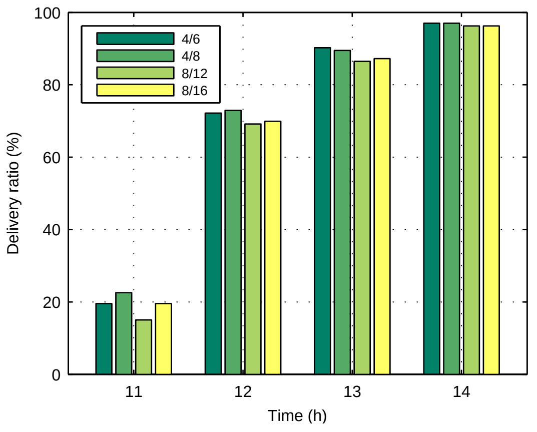
(a) - October 2014 (b) - February 2015

Figure 6.34: MatlabEmulator evaluation impact of the block and generation size during the rush hour period - Percentage of nodes with complete file per hour - Delivery rate

Regarding the delivery delay, the Figure 6.37, Figure 6.38, and Figure 6.39 show that the delivery is faster when the block size is smaller. A more detailed analysis is summarized in Table 6.15, Table 6.16, and Table 6.17. This analysis corroborates the previous statement, being clear the increase of the delivery delay with the block and generation size increment.

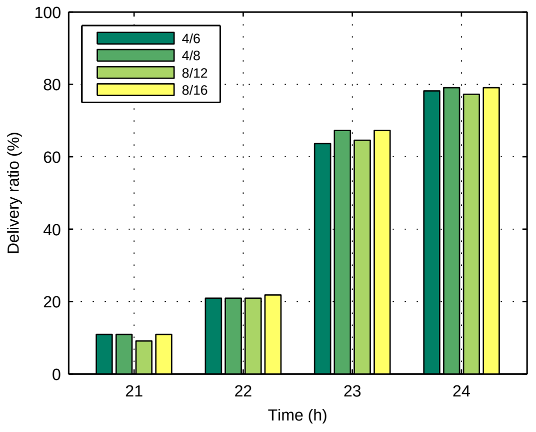


(a) - October 2014

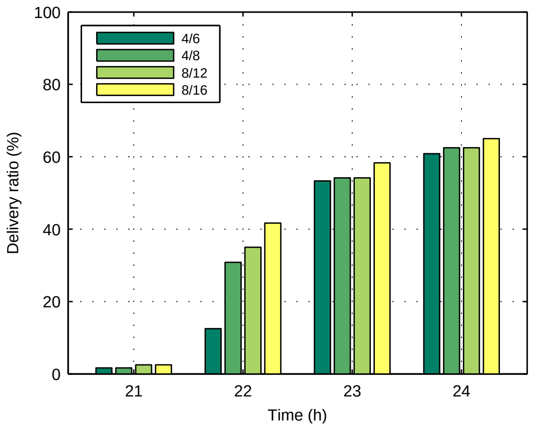


(b) - February 2015

Figure 6.35: MatlabEmulator evaluation impact of the block and generation size during the non rush hour period - Percentage of nodes with complete file per hour - Delivery rate



(a) - October 2014

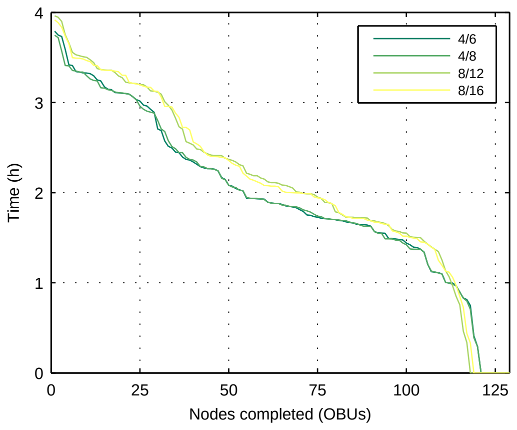


(b) - February 2015

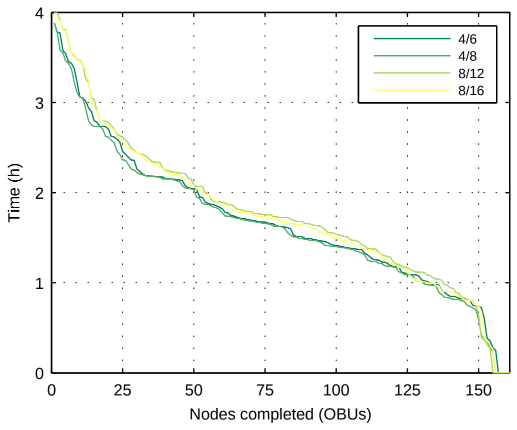
Figure 6.36: MatlabEmulator evaluation impact of the block and generation size during the parking period - Percentage of nodes with complete file per hour - Delivery rate

Block/Gen	**October 2014**		**February 2015**	
	Avg [h]	**C.I. (95%) [h]**	**Avg [h]**	**C.I. (95%) [h]**
4/6	2.10	±0.14	1.74	±0.12
4/8	2.09	±0.14	1.71	±0.12
8/12	2.31	±0.15	1.86	±0.13
8/16	2.29	±0.15	1.82	±0.13

Table 6.15: MatlabEmulator evaluation impact of the block and generation size during the rush hour period - E2E delay statistics

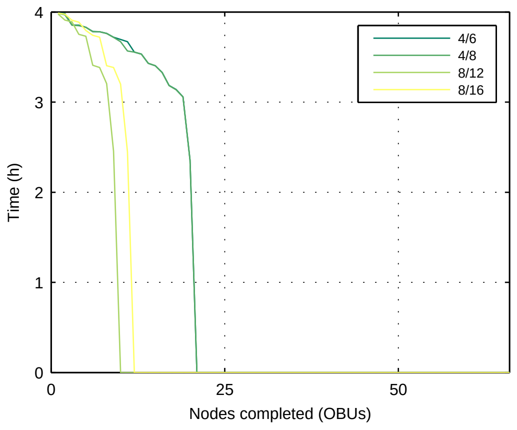


(a) - October 2014

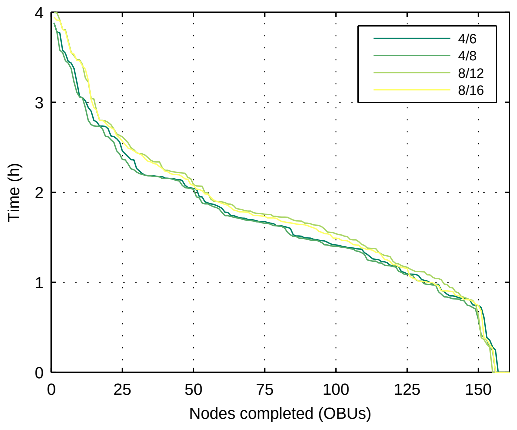


(b) - February 2015

Figure 6.37: MatlabEmulator evaluation impact of the block and generation size during the rush hour period - Time to receive the complete file per node - E2E delay



(a) - October 2014

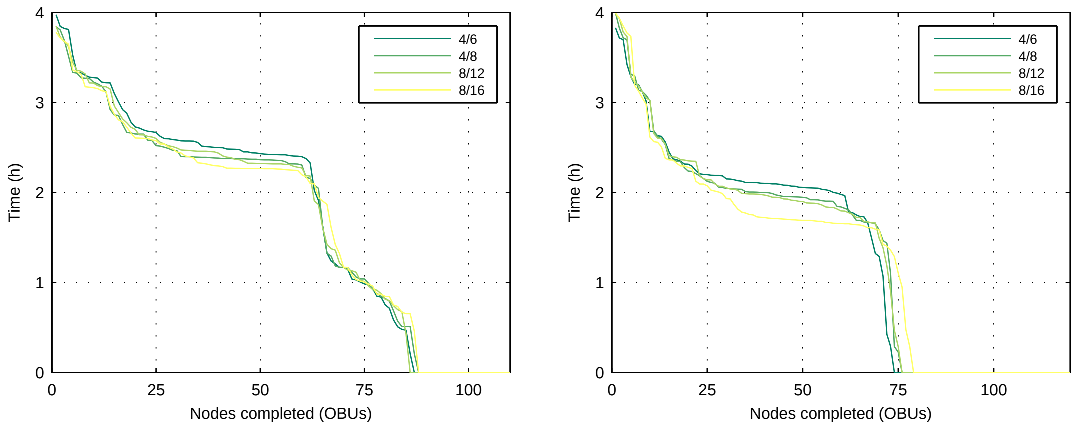


(b) - February 2015

Figure 6.38: MatlabEmulator evaluation impact of the block and generation size during the non rush hour period - Time to receive the complete file per node - E2E delay

Block/Gen	**October 2014**		**February 2015**	
	Avg [h]	**C.I. (95%) [h]**	**Avg [h]**	**C.I. (95%) [h]**
4/6	3.55	±0.18	1.54	±0.14
4/8	3.54	±0.18	1.53	±0.14
8/12	3.52	±0.37	1.69	±0.14
8/16	3.59	±0.31	1.63	±0.14

Table 6.16: MatlabEmulator evaluation impact of the block and generation size during the non rush hour period - E2E delay statistics



(a) - October 2014 (b) - February 2015

Figure 6.39: MatlabEmulator evaluation impact of the block and generation size during the parking period - Time to receive the complete file per node - E2E delay

Block/Gen	**October 2014**		**February 2015**	
	Avg [h]	**C.I. (95%) [h]**	**Avg [h]**	**C.I. (95%) [h]**
4/6	2.27	±0.19	2.19	±0.14
4/8	2.18	±0.18	2.14	±0.15
8/12	2.25	±0.18	2.13	±0.15
8/16	2.18	±0.17	1.99	±0.16

Table 6.17: MatlabEmulator evaluation impact of the block and generation size during the parking period - E2E delay statistics

6.7.2 Strategies to Disseminate Information

After a detailed analysis of the content distribution strategies, it is important to evaluate the proposed techniques to minimize the network congestion as well as increase the delivery rate. Thus, in this subsection three dissemination techniques are analyzed: (i) to optimize delivery, (ii) to minimize the network congestion, and (iii) a hybrid approach trying to achieve both previous goals. These techniques are detailed in section 4.4.

Only the LRBF strategy is evaluated since it displayed the best overall behavior in the previous analysis. Moreover, only the rush hour and parking periods are evaluated since they have higher network congestion whereby the proposed techniques should be applicable to them. The two most important metrics to evaluate are the delivery rate and the network congestion (number of listened packets), whereby only these metrics are evaluated. The file under dissemination is the same as used in all the MatlabEmulator experiment (size of 100 MB).

These techniques are defined according to several parameters as detailed in section 4.4. Thus, the methodology used to evaluate these parameters was a trial and error approach. This might not be the ideal approach, but the usage of different techniques to evaluate these parameters (e.g. fitness function) is not covered in the scope of this Dissertation. The main goal is to identify easily deployed and efficient techniques to minimize the network congestion and achieve a better delivery rate. Due to this objective, once a successful value for these parameters is found, the concept is demonstrated. In the future other approaches can be used in order to evaluate such parameters.

Given that the proposed strategies are used to solve a problem that is detected in the previous analysis – network congestion – and intend to increase delivery rates, the MatlabEmulator will be used as a platform for behavior evaluation. The fact that only the MatlabEmulator is used allows for a quick analysis of the impact of these techniques on content dissemination.

6.7.2.1 Optimize Delivery

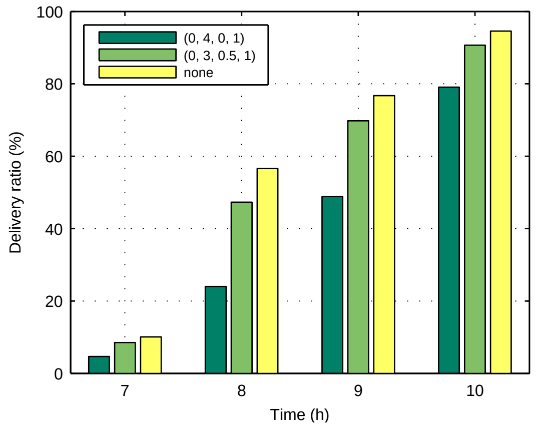
As previously mentioned in section 4.4, in this technique the X_{n_1}, X_{n_2}, P_{n_1}, and P_{n_2} are parameters that characterize the increasing probability function – as illustrated in Figure 4.20 – are defined. The suggested parameters were obtained through trial and error, as mentioned before. As such, two sets of values were defined to be used in this first technique:

- $(X_{n_1}, X_{n_2}, P_{n_1}, P_{n_2}) = (0, 4, 0, 1)$
- $(X_{n_1}, X_{n_2}, P_{n_1}, P_{n_2}) = (0, 3, 0.5, 1)$

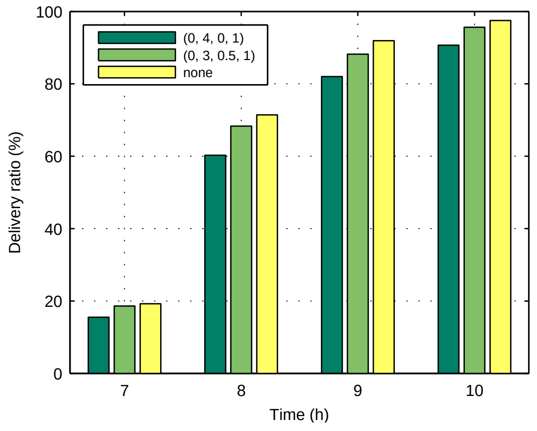
This technique aims to enhance packet delivery by making the decision to send when the sending node has a high number of neighbors instead of cases where there are less neighbors.

Two separate time periods were evaluated in order to analyze the behavior of the suggested technique in these different situations. By analyzing the results obtained for the rush-hour period (see Figure 6.40), it is possible to conclude that the suggested technique does not produce the wanted results. On the other hand, for the parking period (see Figure 6.41) a higher delivery rate is achieved when this technique is used. These results may be justified by the fact that, during the rush-hour period, the average number of valid contacts is relatively low and therefore contact opportunities are wasted. As such, the use of this technique leads to a low delivery rate. The case of the parking period is different given that for most of the time the vehicles are parked in the parking lot whereby they have a high number of neighbors.

As such, by enhancing the delivery when the nodes have a high number of neighbors, it is possible to attain higher delivery rates for most time periods.

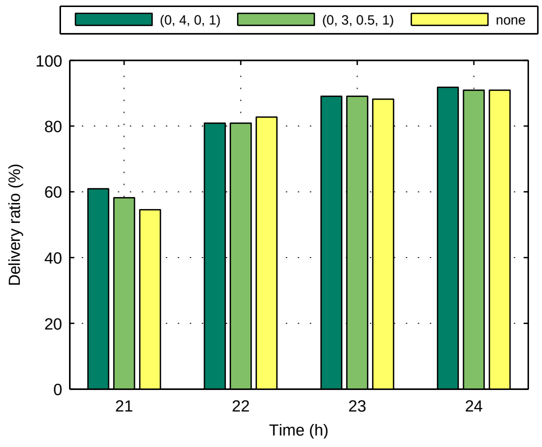


(a) - October 2014

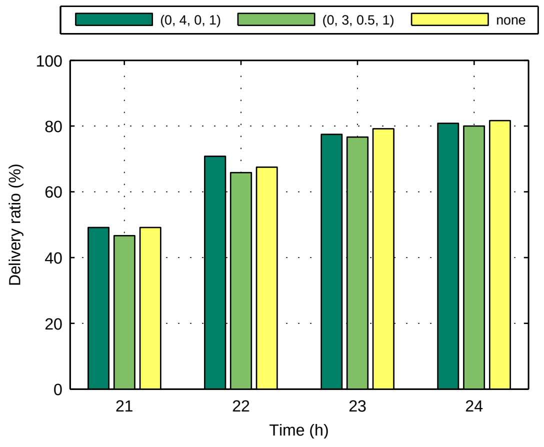


(b) - February 2015

Figure 6.40: MatlabEmulator evaluation of optimize delivery technique in rush hour period - Percentage of nodes with complete file per hour - Delivery rate



(a) - October 2014

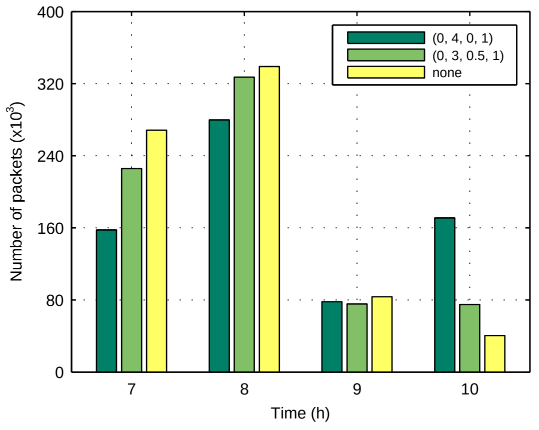


(b) - February 2015

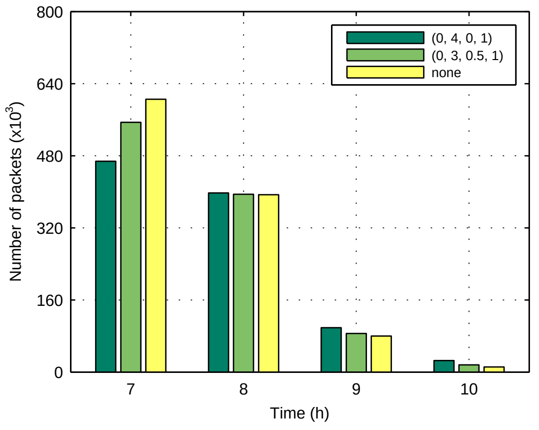
Figure 6.41: MatlabEmulator evaluation of optimize delivery technique in parking period - Percentage of nodes with complete file per hour - Delivery rate

During the rush-hour period it is possible to observe that network congestion (see Figure 6.42 and Figure 6.43) is higher at the beginning of the experiment, when no technique for delivery optimization is used. However, this trend is altered over the course of the experiment. As mentioned before, this behavior can be explained by the fact that in the LRBF strategy, as the delivery rate increases, network congestion decreases since a large number of nodes already has the file, and there is no further need for continuous file sending. During the parking period it is possible to observe that this technique leads to a decrease in network congestion when compared to a situation where this technique is not applied. The reasons that justify this behavior are the same as the ones mentioned for the rush-hour period.

It is therefore possible to conclude that the suggested technique should mainly be applied in time periods and scenarios where the average number of valid neighbors is high. Out of the tested scenarios, this technique proved to be more applicable during the parking period.

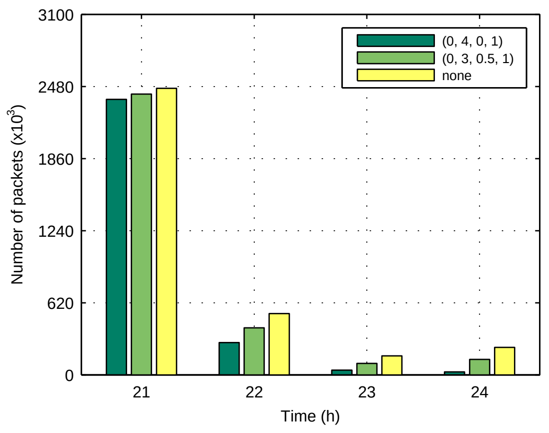


(a) - October 2014

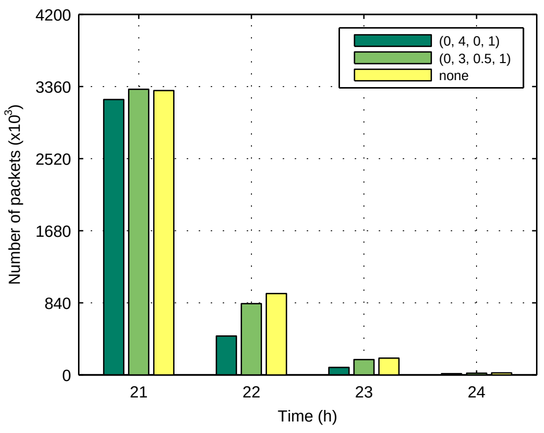


(b) - February 2015

Figure 6.42: MatlabEmulator evaluation of optimize delivery technique in rush hour period - Number of listened packets by the network (only OBUs) throughout the experiment



(a) - October 2014



(b) - February 2015

Figure 6.43: MatlabEmulator evaluation of optimize delivery technique in parking period - Number of listened packets by the network (only OBUs) throughout the experiment

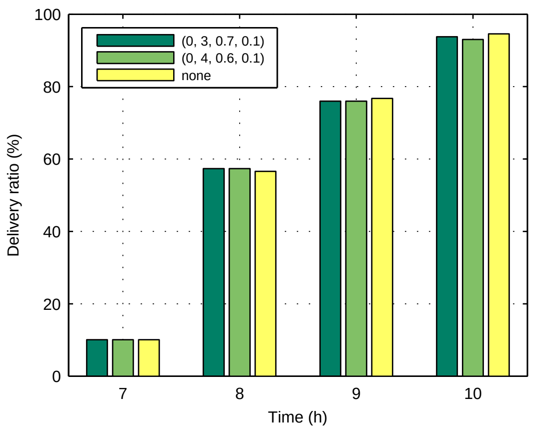
6.7.2.2 Minimize Congestion

Section 4.4 describes this technique, in which the X_{h_1}, X_{h_2}, P_{h_1}, and P_{h_2} are parameters that characterize the decreasing probability function, as illustrated Figure 4.21. Similarly to the previous technique, the parameters are suggested based on a trial and error approach. As such, two sets of values were defined to be used in this second technique:

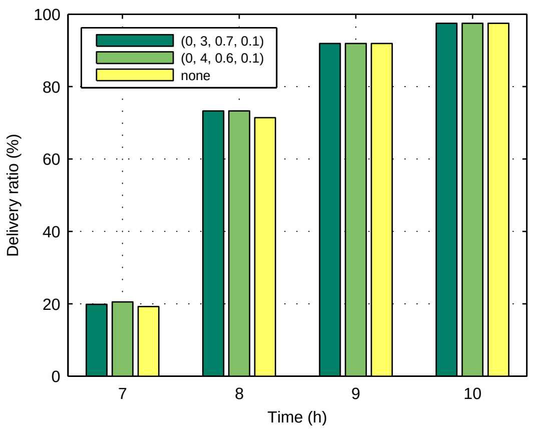
- $(X_{h_1}, X_{h_2}, P_{h_1}, P_{h_2}) = (0, 3, 0.7, 1)$
- $(X_{h_1}, X_{h_2}, P_{h_1}, P_{h_2}) = (0, 4, 0.6, 1)$

The main and only goal of this technique is to reduce network congestion through a limitation based on the number of hops of a certain packet. As such, this technique prioritizes the sending of packets that have a small number of hops, since these traveled less through the network and are therefore probably lacking to a larger number of vehicles. On the other hand, the probability of sending is reduced when a certain packet has a high number of hops since there is a higher probability of the packet existing in a larger number of nodes. Two separate time periods were evaluated in order to analyze the behavior of the suggested technique in these different situations.

The results obtained for the rush-hour period (see Figure 6.44) and for the parking period (see Figure 6.45) are very similar. In both periods the delivery rate slightly increases when the congestion minimization technique is used. This result is justifiable given that a decrease in network congestion leads to an increase in available bandwidth. Therefore, the now-available bandwidth can be used for the sending of the least traveled packets, that exist in a smaller number of vehicles, increasing the diversity of the network packets and enabling a higher delivery rate.



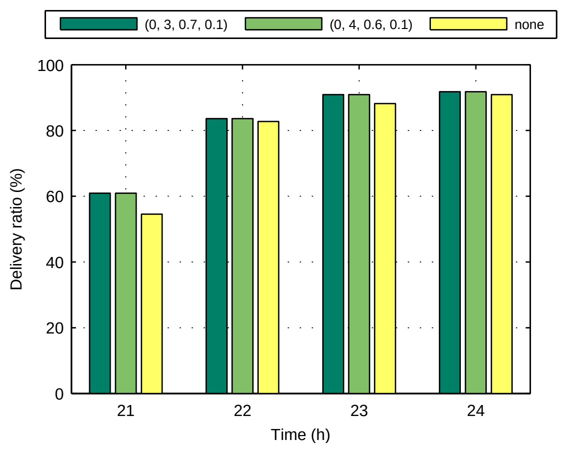
(a) - October 2014



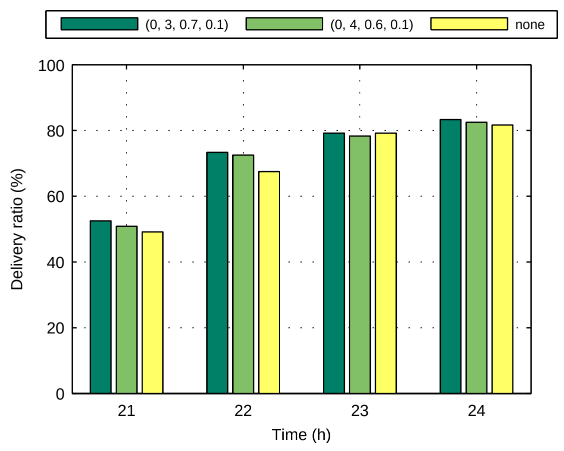
(b) - February 2015

Figure 6.44: MatlabEmulator evaluation of minimize congestion technique in rush hour period - Percentage of nodes with complete file per hour - Delivery rate

Regarding the most important metric associated with this technique, it is possible to conclude that the network congestion is reduced when this technique is used. Both for the rush-hour period (see Figure 6.46) and the parking period (see Figure 6.47) there is a decrease in the number of listened packets in the network. This is justified by the lack of need to re-send highly traveled packets, given that these are probably already highly disseminated in the



(a) - October 2014

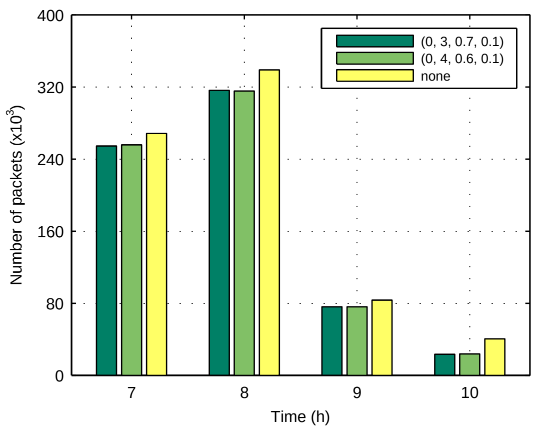


(b) - February 2015

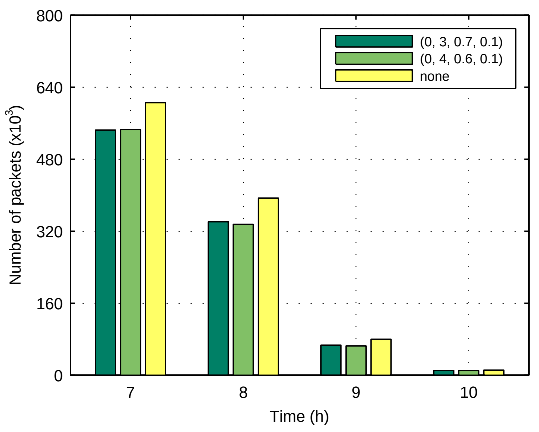
Figure 6.45: MatlabEmulator evaluation of minimize congestion technique in parking period - Percentage of nodes with complete file per hour - Delivery rate

network. This is very relevant during the parking period since in this scenario the majority of the nodes are concentrated in a small geographical area, and therefore do not need to travel much to be disseminated (by broadcast) by a large number of vehicles.

Thus, it is possible to conclude that the proposed technique should mainly be applied to scenarios that are characterized by high vehicle density, such as the parking scenario.

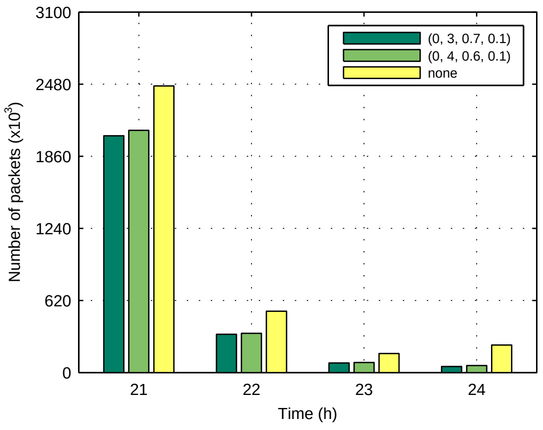


(a) - October 2014

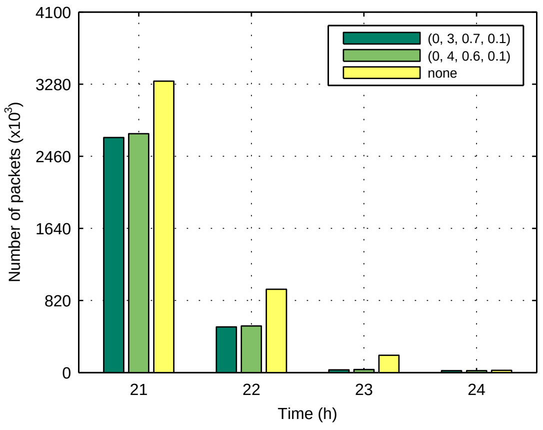


(b) - February 2015

Figure 6.46: MatlabEmulator evaluation of minimize congestion technique in rush hour period - Number of listened packets by the network (only OBUs) throughout the experiment



(a) - October 2014



(b) - February 2015

Figure 6.47: MatlabEmulator evaluation of minimize congestion technique in parking period - Number of listened packets by the network (only OBUs) throughout the experiment

6.7.2.3 Hybrid Approach

Section 4.4 describes this technique, in which the X_{n_1}, X_{n_2}, P_{n_1}, and P_{n_2} are parameters that characterize the increasing probability function, which is multiplied by a decreasing probability function characterized by parameters X_{h_1}, X_{h_2}, P_{h_1}, and P_{h_2}, as illustrated in Figure 4.23. The parameters were defined based on the two previous subsections which already evaluated four sets of values for those parameters. As such, four sets of values were defined to be used in this third technique:

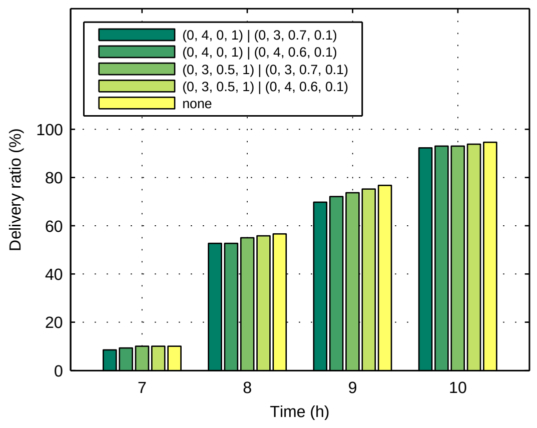
- $(X_{n_1}, X_{n_2}, P_{n_1}, P_{n_2}) = (0, 4, 0, 1)$ and $(X_{h_1}, X_{h_2}, P_{h_1}, P_{h_2}) = (0, 3, 0.7, 0.1)$
- $(X_{n_1}, X_{n_2}, P_{n_1}, P_{n_2}) = (0, 4, 0, 1)$ and $(X_{h_1}, X_{h_2}, P_{h_1}, P_{h_2}) = (0, 4, 0.6, 0.1)$
- $(X_{n_1}, X_{n_2}, P_{n_1}, P_{n_2}) = (0, 3, 0.5, 1)$ and $(X_{h_1}, X_{h_2}, P_{h_1}, P_{h_2}) = (0, 3, 0.7, 0.1)$
- $(X_{n_1}, X_{n_2}, P_{n_1}, P_{n_2}) = (0, 3, 0.5, 1)$ and $(X_{h_1}, X_{h_2}, P_{h_1}, P_{h_2}) = (0, 4, 0.6, 0.1)$

The goal of this technique is to optimize the delivery (making the decision to send when the sending node has a high number of neighbors instead of cases where there are less neighbors), keeping network congestion low (through a limitation based on the number of hops of a certain packet). As such, this technique follows the same premises as the previous two and is focused on taking advantage of their synergy. Thus, the probability of forwarding a packet is lower when the node has a lower number of hops and wants to send a packet which has a high number of hops. On the other hand, the forwarding probability is high if the number of hops is closer to zero and the number of neighbors is high. Two separate time periods were evaluated in order to analyze the behavior of the suggested technique in these different situations.

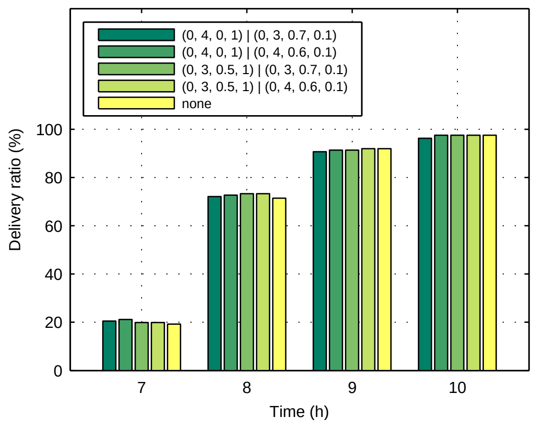
The results obtained for the rush-hour period (see Figure 6.48) and for the parking period (see Figure 6.49) are similar. In both periods, the delivery rate increased when this hybrid technique was applied. These results are justified by the synergy of the two previous approaches, which takes advantage of the gains introduced by each one of them. The previously detected negative effect of the optimized delivery strategy in the delivery rate is evidenced in the rush-hour period of the October dataset.

By analyzing the metric related with the network congestion, it is possible to conclude that it is reduced when this technique is used. Both in the rush-hour period (see Figure 6.50) and the parking period (see Figure 6.51), the number of listened packets in the network decreases. Due to the weight of the minimize congestion strategy, this behavior is more relevant during the parking period, since in this period the majority of the nodes are concentrated in parking lots (which have a restricted area), and therefore do not need to be forwarded through a large number of vehicles.

Thus, it is possible to conclude that the proposed technique should mainly be applied to scenarios which are characterized by high vehicle density, such as the parking scenario.

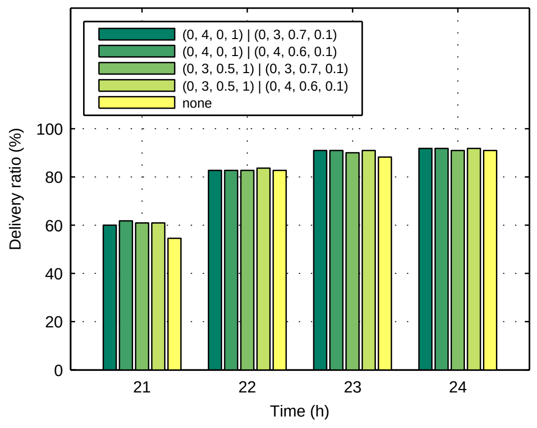


(a) - October 2014

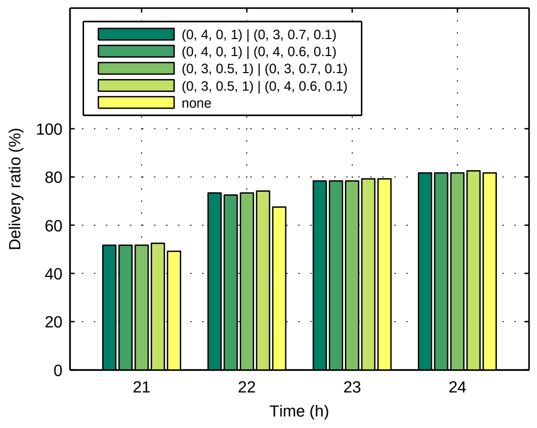


(b) - February 2015

Figure 6.48: MatlabEmulator evaluation of a hybrid approach technique in rush hour period - Percentage of nodes with complete file per hour - Delivery rate

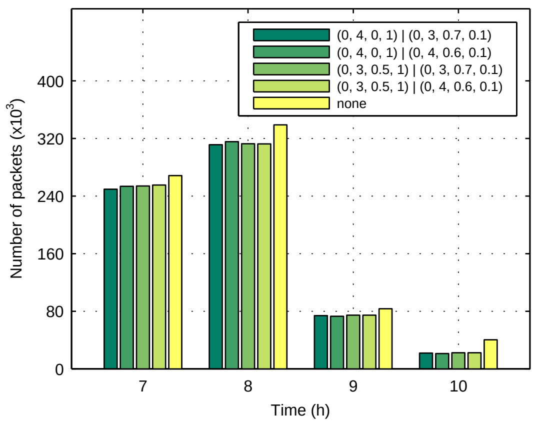


(a) - October 2014

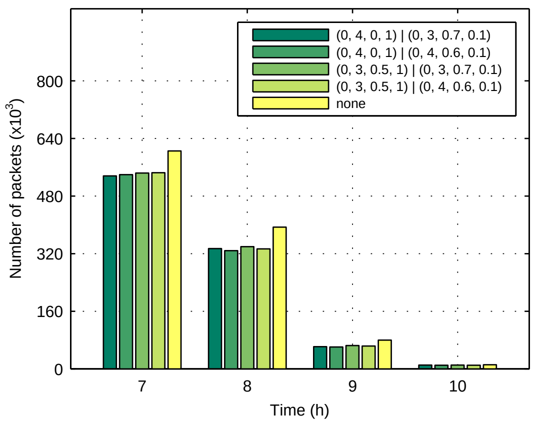


(b) - February 2015

Figure 6.49: MatlabEmulator evaluation of a hybrid approach technique in parking period - Percentage of nodes with complete file per hour - Delivery rate

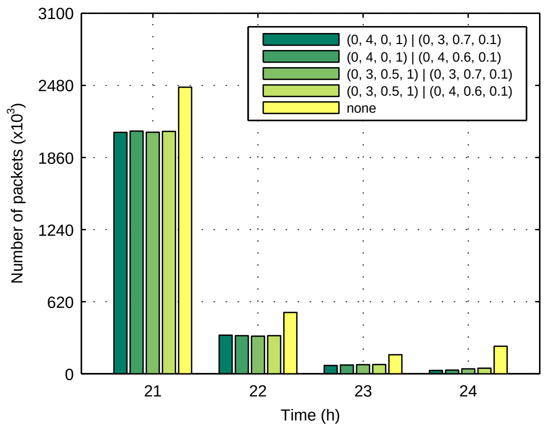


(a) - October 2014

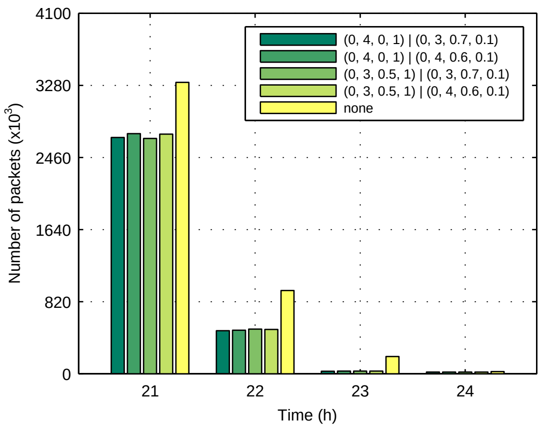


(b) - February 2015

Figure 6.50: MatlabEmulator evaluation of a hybrid approach technique in rush hour period - Number of listened packets by the network (only OBUs) throughout the experiment



(a) - October 2014



(b) - February 2015

Figure 6.51: MatlabEmulator evaluation of a hybrid approach technique in parking period - Number of listened packets by the network (only OBUs) throughout the experiment

6.8 HelixEmulator Evaluation

Several strategies were implemented and evaluated in the MatlabEmulator since it is a simpler platform that allows an easy and fast implementation along with an accurate evaluation. Once these strategies were evaluated, three of them were selected as the ones that should be implemented in the Helix software. Although the proposed strategies (Random, LNHF, LRBF, and LRGF) were implemented, only three of them are evaluated since they are the ones with better performance (in the MatlabEmulator). Thus, in this section the three evaluated strategies are the following: LNHF, LRBF, and LRGF.

The MatlabEmulator and HelixEmulator were designed for the same purpose (evaluate content distribution strategies) but they have different implementations and assumptions. This fact does not allow a complete and direct comparison between their results

The standard experiment is characterized by the dissemination of a 75 MB file divided in 2256 packets of 32 KB each. The disseminated file is smaller compared to the MatlabEmulator evaluation, since the computational capability of the machine used to run the HelixEmulator is not good enough to handle the requirements needed to disseminate a larger file. The software used to create and run a VM does not allow the extension of the processing power (increase number of cores or their frequency), whereby this extension was not possible. Only two of the three previously described scenarios are evaluated: (i) Rush Hour and (ii) non-Rush Hour.

This emulator introduces the broadcasting of advertisement messages presenting a more closer behavior to a real vehicular environment. Thus, the advertisement messages are used by a node to discover the storage content of its vicinity. Moreover, in this emulator it is possible to evaluate the overhead introduced by the dissemination of this type of meta-data and its impact in the network.

In order to approximate the emulator behavior to the real behavior of this type of service in the real network the bandwidth is limited to 1 Mbps using the BandwidthLim module. Similarly to the MatlabEmulator, only the second situation of the LRGF strategy (see section 4.3) is evaluated. In this strategy, a block is composed by 8 packets and, when coded, it results in a generation of 12 coded packets since these are the most common values in the literature.

In order to minimize the performance problems of the HelixEmulator several periodical procedures do not have a fixed period, whereby they are defined as a random value in a range. Thus, the advertisement packets are broadcasted with a periodicity between 5 and 10 seconds, and the refreshing of the internal structures between 15 and 30 seconds. These are the default values. The valid time of an internal structure information, received from an advertisement packet, is restricted to 22 seconds.

6.8.1 Strategies to Stateless Choose Information

6.8.1.1 Rush Hour Period

Figure 6.52 and Figure 6.53 illustrate the evolution of metrics associated with the percentage of the delivery throughout the experiment in the rush hour period. In the first one only the nodes which completely receive the file are taken into account, and the other one is related to the percentage of file downloaded in the network (even if the content is not completed).

The results clearly show that LRBF and LRGF have the best behavior in terms of delivery, although the LRGF takes more time than LRBF to achieve the same delivery rate. Contrarily, the LNHF strategy can not completely deliver the file, whereby it presents a lower efficiency compared to the other two.

In terms of the percentage of file download by the network nodes, all strategies converge to a high percentage. However, the LNHF strategy takes more time than the other two strategies to achieve the same percentage. Thus, it can be concluded that the intelligence introduced in the LRBF and LRGF strategies can lead to a better performance regarding a higher delivery rate with lower delay.

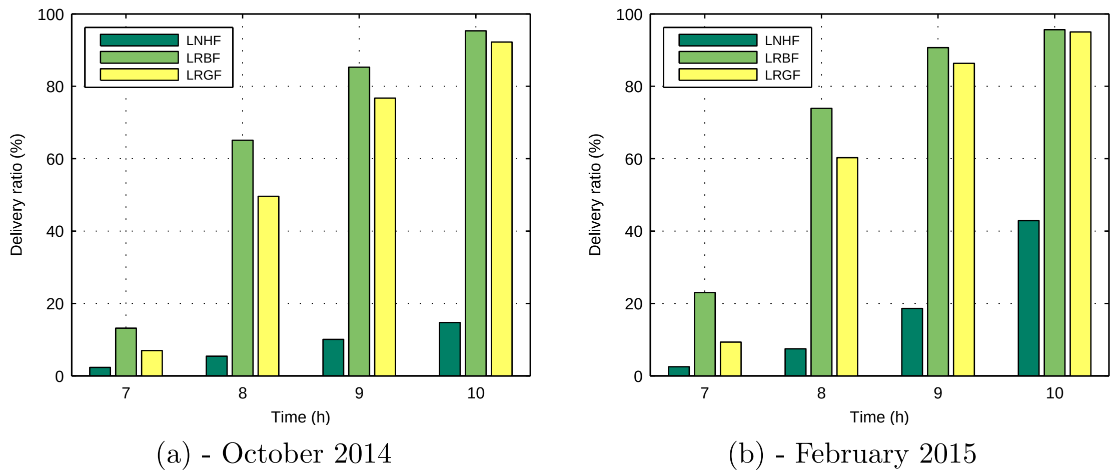


Figure 6.52: HelixEmulator evaluation comparison in rush hour period - Percentage of nodes with complete file per hour - Delivery rate

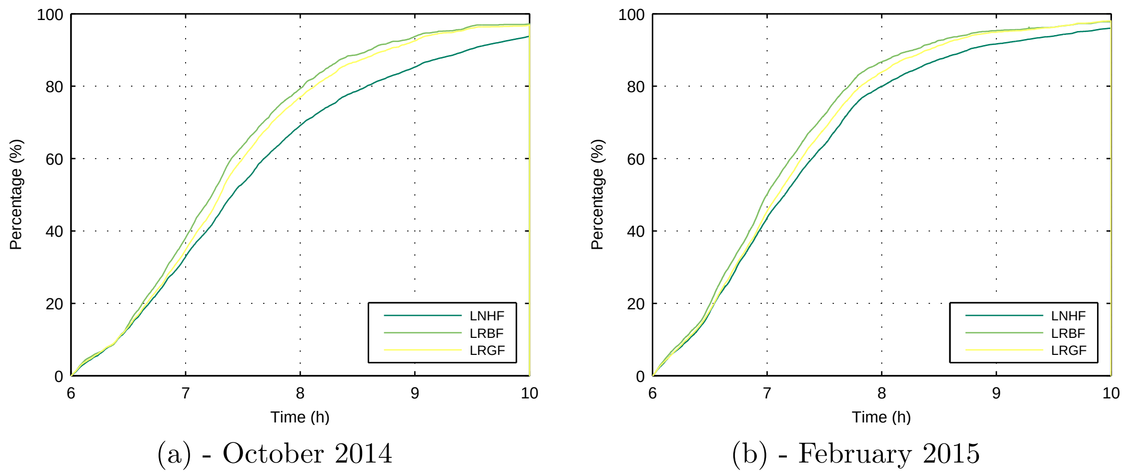
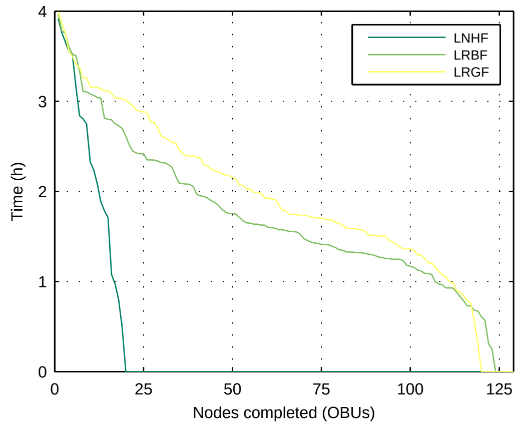


Figure 6.53: HelixEmulator evaluation comparison in rush hour period - Percentage of file distributed in network throughout the experiment

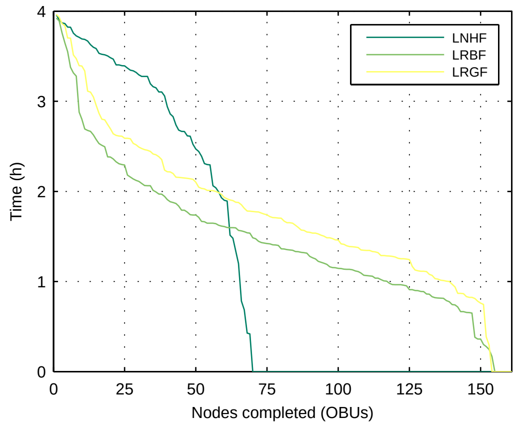
Figure 6.54 represents the time that each node takes to download the complete content, whereby only information about nodes which completely downloaded the file are displayed. Figure 6.55 is the progress rate and aims to give a perspective about how quick the delivery is.

According to Table 6.18 and analyzing the results, it is possible to concluded that the

LRBF and LRGF strategies have the best performance since, in mean, the download of the content is faster than in the LNHF strategy. In addition to that, as the progress rate shows, in the two better strategies (LRBF and LRGF) the great majority of downloads finish in the first two hours. Further, in the LNHF strategy this occurs during the last half of the experiment.

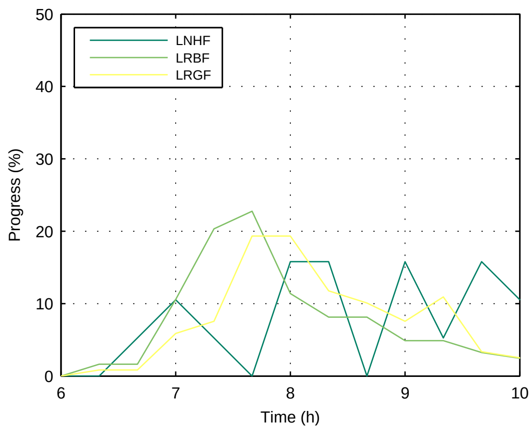


(a) - October 2014

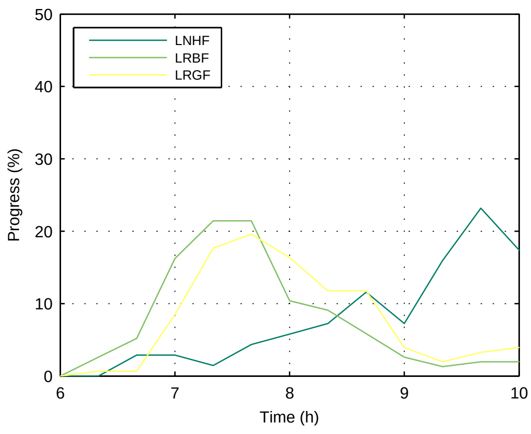


(b) - February 2015

Figure 6.54: HelixEmulator evaluation comparison in rush hour period - Time to receive the complete file per node - E2E delay



(a) - October 2014



(b) - February 2015

Figure 6.55: HelixEmulator evaluation comparison in rush hour period - Progress rate

The metric illustrated in Figure 6.56 regards the analysis of the network congestion. Similarly to the MatlabEmulator evaluation, when an OBU hears a packet, a counter is incremented. Thus, if this value is higher in a certain strategy than in another, it means that the medium is more congested.

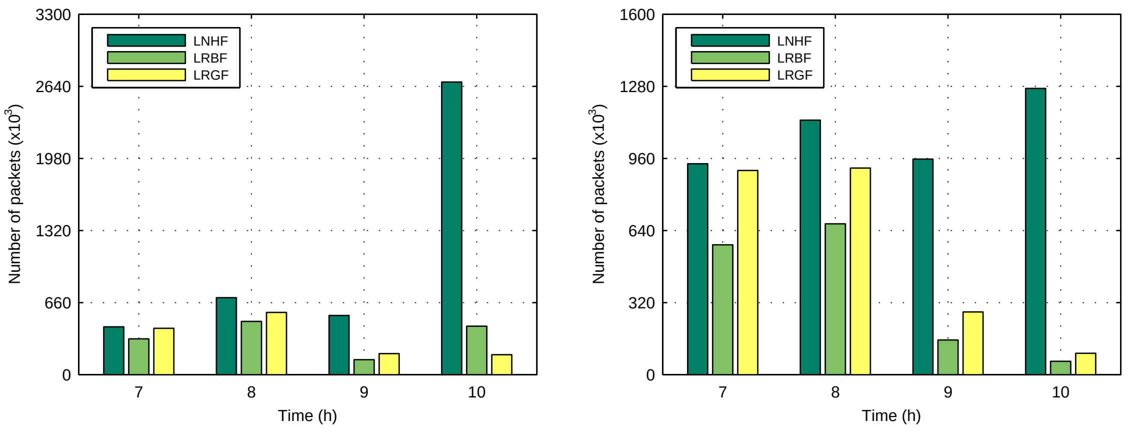
As in the MatlabEmulator evaluation, the results of the two datasets are very different. According to the initial study of the network, in the first dataset (October) there is a high increase in the number of contacts between 9am and 10pm which justified the increase of listened packets in this period. On the other hand, the number of contacts in February is

Strategy	October 2014		February 2015	
	Avg [h]	C.I. (95%) [h]	Avg [h]	C.I. (95%) [h]
LNHF	2.39	±0.52	2.86	±0.21
LRBF	1.77	±0.14	1.54	±0.12
LRGF	2.06	±0.14	1.85	±0.12

Table 6.18: HelixEmulator evaluation comparison in rush hour period - E2E delay statistics

more stable whereby the number of listened packets in the LNHF are approximately equal and constant throughout the experiment. The sudden increase is only verified in the LNHF strategy due to its lack of intelligence; since if it has more valid contacts (opportunities to transmit packets), it sends a higher number of packets.

In both datasets a decrease in the number of listened packets during the last periods of the experiment when the LRBF or LRGF are used is registered. This behavior is related to the fact that the transmission of packets decreases when the majority of a node's neighbors have successfully downloaded the content. In this case a node does not send any data packet once no one in its vicinity needs it. The LRBF has the lowest number of listened packets since it has the highest delivery ratio and lowest E2E delay.



(a) - October 2014 (b) - February 2015

Figure 6.56: HelixEmulator evaluation comparison in rush hour period - Number of listened packets by the network (only OBUs) throughout the experiment

The set of metrics that are illustrated in Figure 6.57 and Figure 6.58 have the goal of analyzing the impact of advertising packets on the network.

An increase in the number of advertising packets sent is visible over time. This was expected since the more nodes are aware of the content under dissemination, the larger the number of advertisements in the network.

In terms of the size of advertisements, there is a clear difference between both strategies. At the start of the process, the LRBF strategy adds a larger overhead to the network when compared to the LRGF strategy. This is due to the fact that in LRBF, while the node has not yet received the entire file, it will be announcing every hash it owns of that same file (in order to map every package of the file 4 bytes are needed). Once it possesses the entire file, the node proceeds to only send generic information about it (total number of packets and identifier), which largely reduces the size of the advertisement packets. On the other hand, the size of the advertisement packets in the LRGF strategy has smaller variance given that, in the worst case scenario, they send the ranks of all the generations (total number of generations is less than the total number of packets) and, in the best case, they only send generic information about the content (file identifier, block size, generation size, etc.).

When both strategies converge in terms of delivery rates, it is expect that, in the final period of the experiment, the total size of the advertisement packets is higher in LRGF due to the aforementioned reasons.

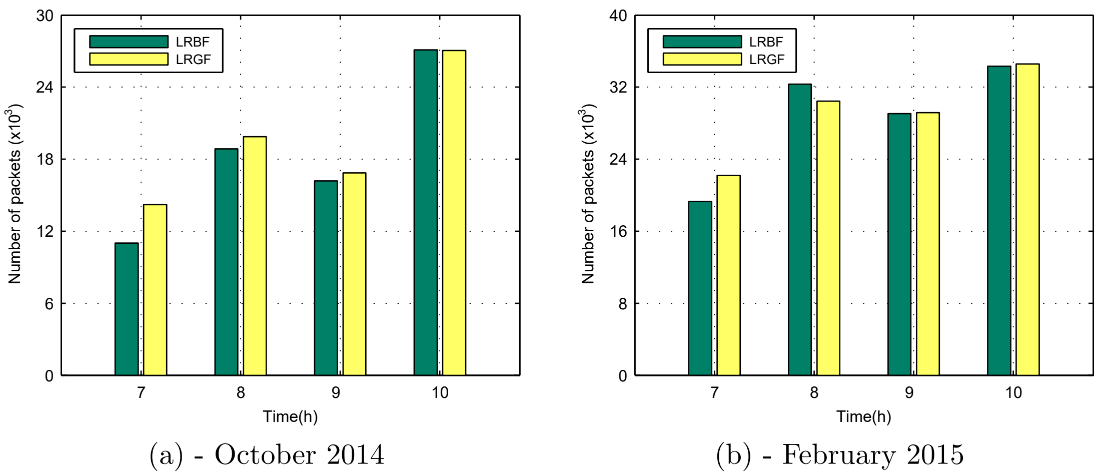


Figure 6.57: HelixEmulator evaluation comparison in rush hour period - Number of transmitted advertisement packets in network throughout the experiment

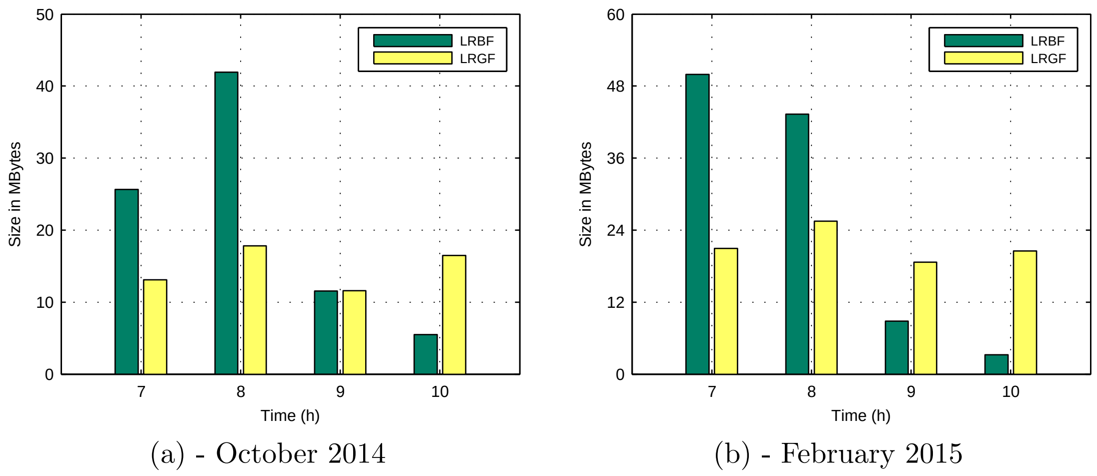


Figure 6.58: HelixEmulator evaluation comparison in rush hour period - Size of transmitted advertisement packets in network throughout the experiment

Finally, Table 6.19 resumes the performance of the HelixEmulator machine. Three metrics are monitored and all of them confirm the correct behavior of the machine. The mean load is below the maximum value (which is 7 since it is the number of cores minus one).

	Strategy	CPU [%]		Load		Memory [%]	
		Avg	Std	Avg	Std	Avg	Std
Oct 2014	LNHF	10.3	4.7	2.2	2.8	12.3	4.0
	LRBF	14.2	12.0	2.0	2.4	28.1	1.9
	LRGF	5.3	2.8	0.7	0.8	10.5	2.5
Feb 2015	LNHF	7.0	1.6	1.1	1.3	16.9	6.1
	LRBF	9.1	11.9	1.3	1.8	23.6	5.6
	LRGF	3.6	1.9	0.5	0.9	31.8	2.2

Table 6.19: HelixEmulator evaluation comparison in rush hour period - Computational performance statistics

6.8.1.2 non-Rush Hour Period

This scenario has a very different profile comparing to the last one since there are less vehicles and consequently a lower number of contacts. This behavior is more pronounced in October than in February due to the addition of new RSUs which leads to a better network coverage.

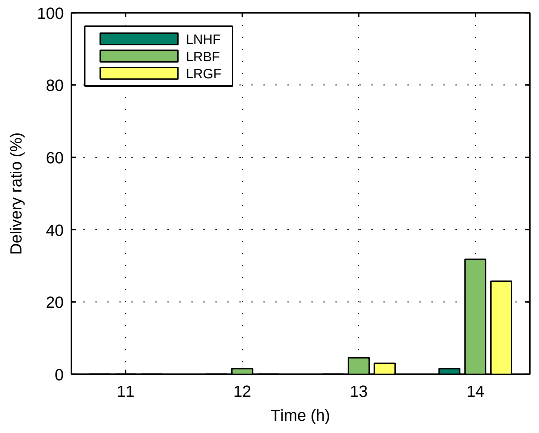
The following analysis aims to evaluate the performance of the three strategies (previously analyzed) in this period. The analysis is structured as the previous one, being focused on the delivery ratio, time concerning, network congestion, impact of advertisement packets, and computational performance.

Figure 6.59 and Figure 6.60 show that the delivery rate in the October dataset is very low. Similarly to the MatlabEmulator evaluation, this results from a lack of connectivity during the trajectory from the city to the parking lot (where vehicles are parked during the midday). Due to the arrival of a large number of vehicles from the parking lots to the city center, the delivery rate abruptly increases in the last hour of the experiment (in October). However, when analyzing the delivery rate in February, the effect of the network reinforcement (increase in number of RSUs and network coverage) is clear. During the traveling period from the parking lots to the city center, the vehicles pass through a higher number of RSUs, enhancing the delivery.

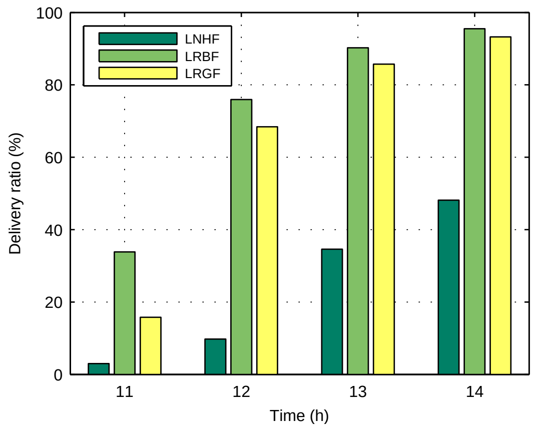
Comparing to the rush hour period, the evolution of the delivery rate throughout the experiment remains the same, whereby the LRBF presents the highest delivery rate, followed by the LRGF, and with a very low delivery rate there is the LNHF strategy.

Regarding the delivery delay in the October dataset, the majority of the nodes finish the content download in the final hour due to reasons stated before. On the other hand, in the February dataset, the behavior is similar to the one registered during the rush hour period. Moreover, as Table 6.20 shows, the strategies with better performance are the LRBF and LRGF strategies, followed by the LNHF strategy. Figure 6.61 and Figure 6.62 confirm the previous statement.

Figure 6.63 illustrates the metric related to the network congestion. Comparing the two datasets, a different behavior from October to February is detected. In October, between 10am and 12am, the number of contacts is high which leads to a higher number of listened packets. In the end of the experiment these metrics decrease drastically due to the lower number of contacts. Due to the flat number of contacts in February, the number of listened packets is more constant (at least in LNHF strategy). Nevertheless, both periods registered

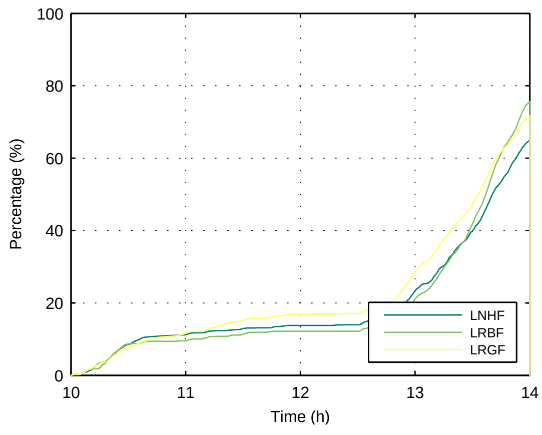


(a) - October 2014

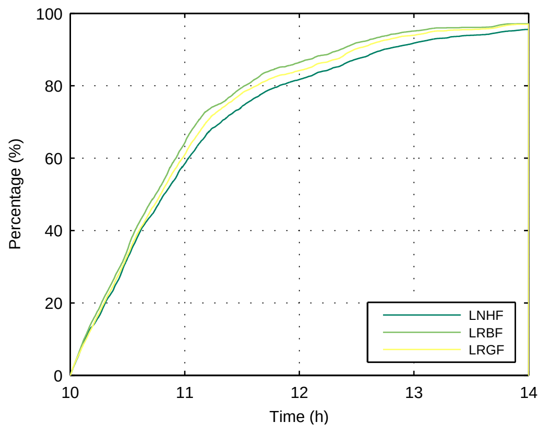


(b) - February 2015

Figure 6.59: HelixEmulator evaluation comparison in non rush hour period - Percentage of nodes with complete file per hour - Delivery rate



(a) - October 2014



(b) - February 2015

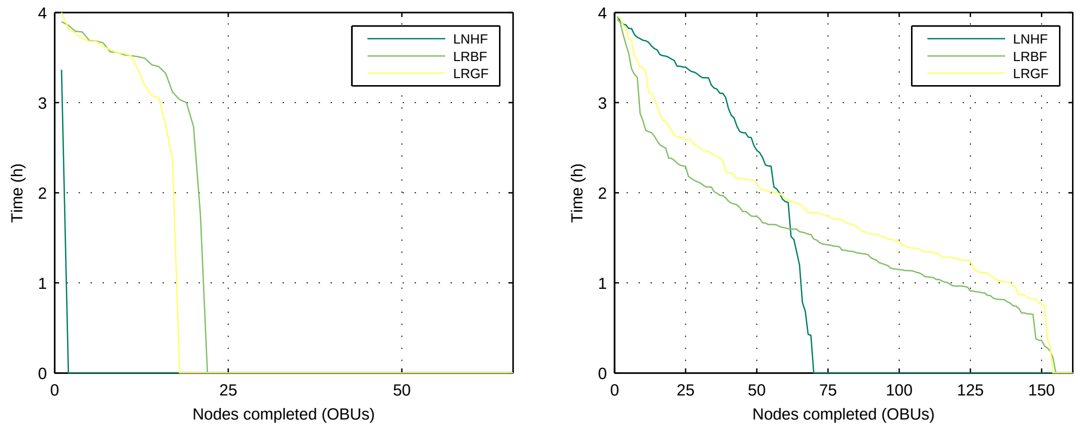
Figure 6.60: HelixEmulator evaluation comparison in non rush hour period - Percentage of file distributed in network throughout the experiment

Strategy	**October 2014**		**February 2015**	
	Avg [h]	**C.I. (95%) [h]**	**Avg [h]**	**C.I. (95%) [h]**
LNHF	3.37	-	2.53	±0.19
LRBF	3.40	±0.22	1.36	±0.13
LRGF	3.43	±0.22	1.66	±0.14

Table 6.20: MatlabEmulator evaluation comparison in non rush hour period - E2E delay statistics

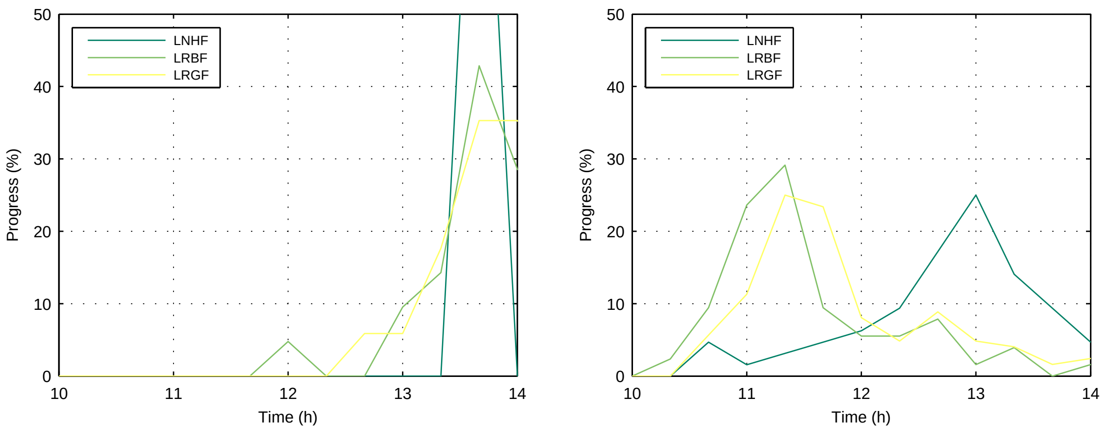
a gradual decrease of listened packets in the LRBF and LRGF strategies. This fact is due to the intelligence of these strategies which improves the transmission decision.

The number of listened packets in the LNHF strategy varies according to the number of



(a) - October 2014 (b) - February 2015

Figure 6.61: HelixEmulator evaluation comparison in non rush hour period - Time to receive the complete file per node - E2E delay



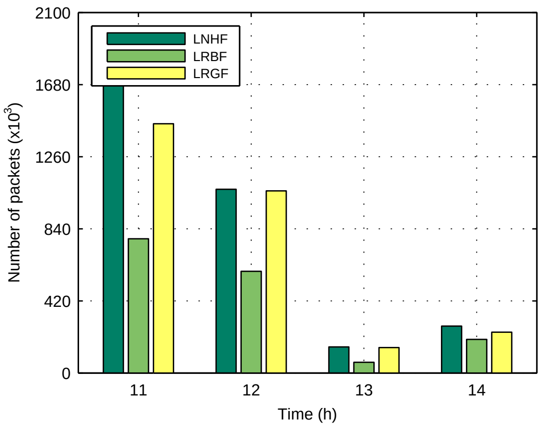
(a) - October 2014 (b) - February 2015

Figure 6.62: HelixEmulator evaluation comparison in non rush hour period - Progress rate

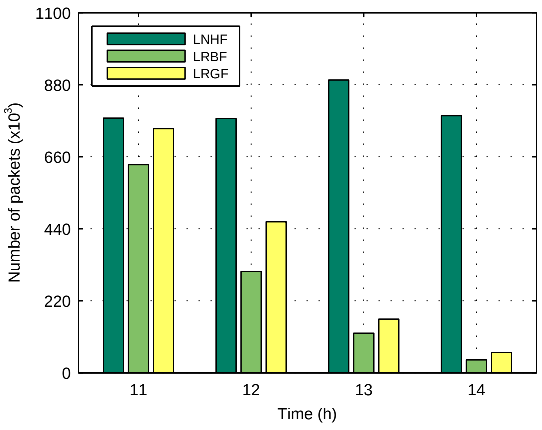
contacts, whereby it remains stable in the periods where the number of contacts is flat. On the other hand, the number of listened packets decreases gradually in the LRBF and LRGF, since in these strategies the nodes only send packets if their vicinity needs additional data. Thus, the number of listened packets decreases with the increase of delivery rate.

Figure 6.64 and Figure 6.65 illustrate metrics that relate to additional advertisement information in the network. These metrics are useful for analyzing the impact of this type of information in the network.

Regarding the October dataset, the number of advertisement packets is higher in the initial and final periods, and there is a decrease in the period between 1pm and 2pm. This behavior may be due to the fact that the number of contacts – and, therefore, the number of neighbors to whom information can be transmitted – is relatively low in the period between 1pm and 2pm (vehicles are parked). The increase in the number of advertisement packets in the period after 2pm is due to the fact that the nodes move from the parking lots to the city



(a) - October 2014



(b) - February 2015

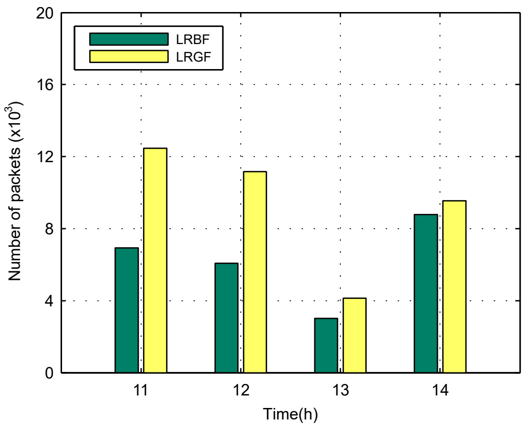
Figure 6.63: HelixEmulator evaluation comparison in non rush hour period - Number of listened packets by the network (only OBUs) throughout the experiment

center, which means that more RSUs are encountered. As such, it is possible to download content and initiate the advertisement broadcast process. In February, the behavior is much more uniform given that the increase in network coverage that occurred minimized the dark zones between the parking lots and the city center.

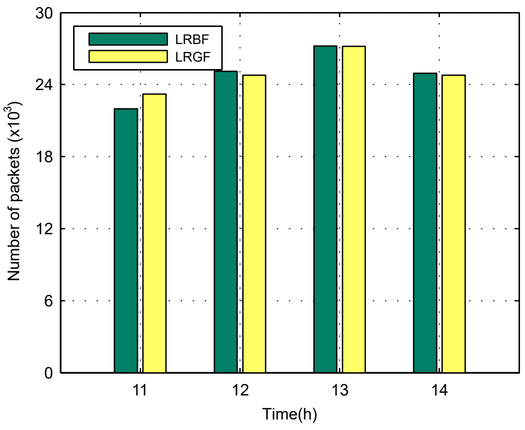
When analyzing the size of the advertisement packets that were disseminated in each strategy, it is possible to conclude that the behavior in February is very similar to the one observed during the rush hour period. As such, and for the same reasons given for the rush hour period (type and size of announced information), it is possible to observe that, at the start of the process, the LRBF strategy adds a larger overhead to the network when compared to the LRGF strategy. At the end of the experiment and considering that the strategies converged in terms of delivery rates, the total size of the advertisement packets is larger in the LRGF strategy due to the previously mentioned reasons regarding the rush hour period.

Regarding the October dataset, the reduced delivery rate and the low percentage of file dissemination in the network do not allow to support the conclusions related to the added overhead in the initial part of the experiment. However, it is possible to analyze the last hour of this experiment, since there was a considerable increase in the delivery rate and the dissemination process was able to deliver a significant amount of packets to the network. In this last hour the behavior is similar to the first hour of the experiment with the February dataset. Thus, it is possible to observe that the added overhead by the LRBF is higher than that of the LRGF strategy.

Finally, Table 6.21 resumes the performance of the HelixEmulator machine in the non-rush hour period. Similarly to the previous analysis, three metrics are monitored and all of them confirm the correct behavior of the machine. Is is shown that the mean load is below the maximum value (7 since it is the number of cores minus one).

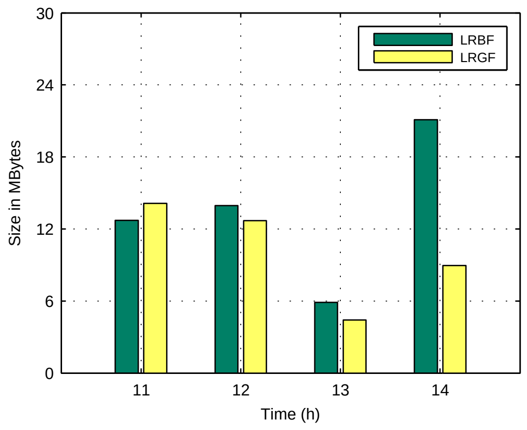


(a) - October 2014

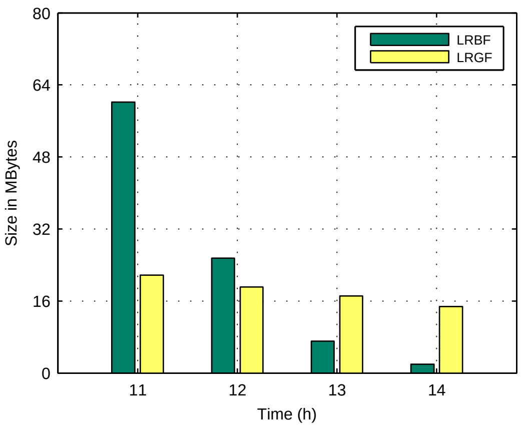


(b) - February 2015

Figure 6.64: HelixEmulator evaluation comparison in non rush hour period - Number of transmitted advertisement packets in network throughout the experiment



(a) - October 2014



(b) - February 2015

Figure 6.65: HelixEmulator evaluation comparison in non rush hour period - Size of transmitted advertisement packets in network throughout the experiment

	Strategy	**CPU [%]**		**Load**		**Memory [%]**	
		Avg	Std	Avg	Std	Avg	Std
Oct 2014	LNHF	3.1	1.3	0.3	0.5	21.4	1.5
	LRBF	5.2	5.0	0.4	0.5	18.6	3.3
	LRGF	5.6	3.7	0.9	1.5	27.6	1.4
Feb 2015	LNHF	5.7	1.0	0.8	0.9	29.3	9.7
	LRBF	7.3	5.5	0.9	1.2	31.0	10.1
	LRGF	2.9	1.4	0.3	0.5	38.3	5.2

Table 6.21: HelixEmulator evaluation comparison in non-rush hour period - Computational performance statistics

6.8.1.3 Parking Period

The parking period is characterized as a time period where the majority of the vehicles were, are, or will be parked in the parking lot. Thus, this is the period when the vehicles density is higher whereby the neighboring list of each node is quite larger than in the other time periods. As previously mentioned (see section 6.8) the software used to create and run a VM does not allow the extension of the processing power (increase number of cores or their frequency).

The combination of the two previous issues/characteristics results in a major challenge for the HelixEmulator machines, responsible for running the emulator. The high number of nodes trying to communicate with other nodes (in broadcast) produces a large number of emulator internal messages (ZMQ messages) for control and data communication. Moreover, the large amount of logging data communicated by the database to the emulator along with the queries represents a huge amount of processing. The specifications of the available machines are not robust enough to handle this level of processing. Several attempts were performed but all of them with the same result: the VM crashes by overload CPU. Since there was no possibility to leverage or improve the VM specifications, this period was not evaluated.

Although it was not evaluated, and based on the similar behavior between the MatlabEmulator and HelixEmulator registered in the previous periods, a similar behavior is expected.

6.8.1.4 Analysing the Impact of Content Distribution Parameters

After a detailed analysis of the content distribution strategies configured as default, it is important to understand what is the impact of some parameters in the dissemination process. Thus, in this subsection several parameters are varied and the content distribution strategies are evaluated.

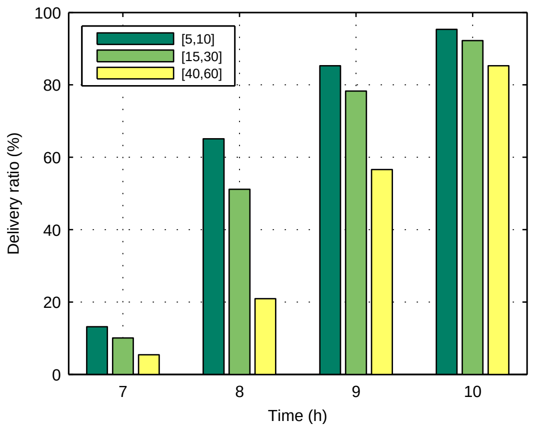
Since the rush hour period was the one with more accurate results (October dataset of non-rush hour period presented a lower delivery rate), this period was selected to evaluate the impact of such parameters. Only the LRBF strategy is evaluated since it had the best overall behavior in the previous analysis. This test is only performed in this section since the advertisement packets and period events related to the internal structures were only added to the HelixEmulator integration (in MatlabEmulator the process was simpler)

Impact of the Advertisement Packets Periodicity

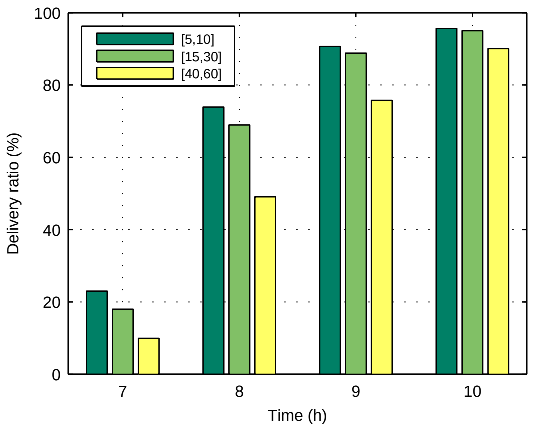
The first analysis aims to evaluate the impact of the advertisement packets' periodicity in the content distribution. As previously mentioned, the advertisement periodicity is not fixed, being framed in a specific range. Thus, three ranges are evaluated: (i) 5 to 10 seconds, (ii) 15 to 30 seconds, and (iii) 40 to 60 seconds. All other periodic events are performed as default.

Figure 6.66 represents the delivery ratio (percentage of nodes with complete file) per hour. Both October and February datasets have the same behavior. Thus, the lower the period of advertisement packets transmission, the higher the delivery ratio. This is an expected result since the faster the advertisement procedures, the faster the update of the internal structures, improving the broadcasting decision. This improved decision allows a faster delivery of the content.

Figure 6.67 illustrates the time to receive the complete file per node, and Table 6.22 summarizes the average delivery delay for all the three evaluated ranges. As expected, when the advertisement packets are sent more often, the delivery delay is lower than in the other ap-



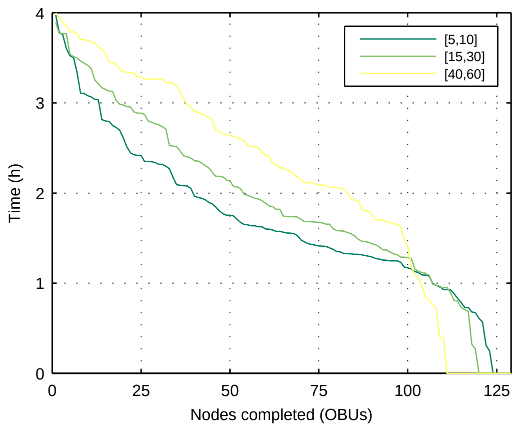
(a) - October 2014



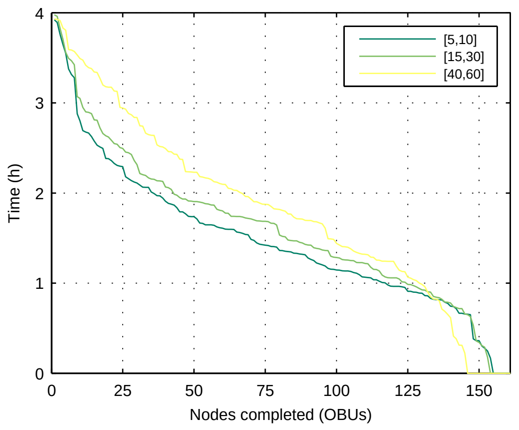
(b) - February 2015

Figure 6.66: HelixEmulator evaluation impact of the advertisement packets periodicity during the rush hour period - Percentage of nodes with complete file per hour - Delivery rate

proaches. As the periodicity increases the delivery delay also increases. Figure 6.68 reinforces the previous statements, since it is clear that the majority of the deliveries occur sooner when the dissemination of advertisement packets is more often.



(a) - October 2014

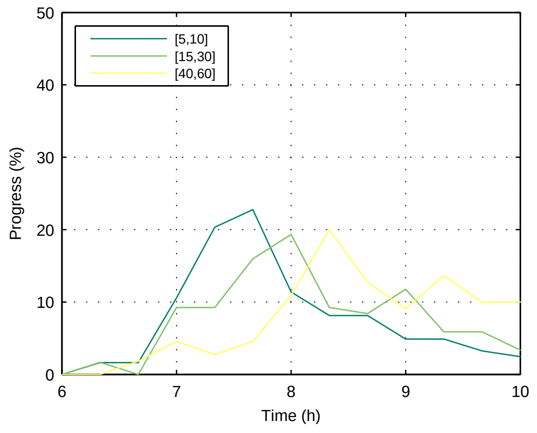


(b) - February 2015

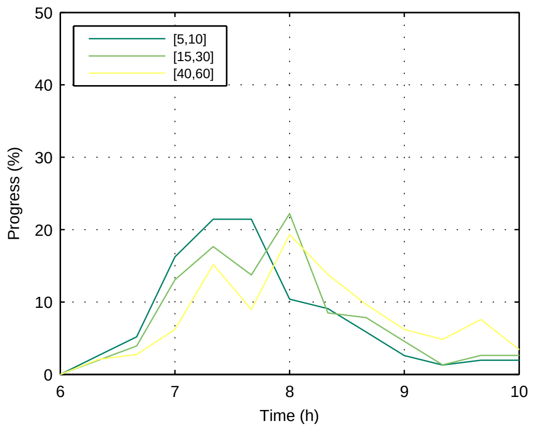
Figure 6.67: HelixEmulator evaluation impact of the advertisement packets periodicity during the rush hour period - Time to receive the complete file per node - E2E delay

As previously mentioned for the LRBF strategy, the network congestion is related to the delivery rate whereby the lower the delivery rate, the potentially higher is the network congestion. Figure 6.69 clearly shows this behavior. At the beginning of the experiment the network congestion is higher when the advertisement periodicity is lower because nodes update their internal structures more often and send more packets to their vicinity. However, as the nodes receive the content the network congestion tends to decrease. Since this delivery is lower when the dissemination of the advertisement packets is less often, the network congestion increases with the advertisement periodicity.

Figure 6.70 and Figure 6.71 represents the number of transmitted advertisement packets



(a) - October 2014

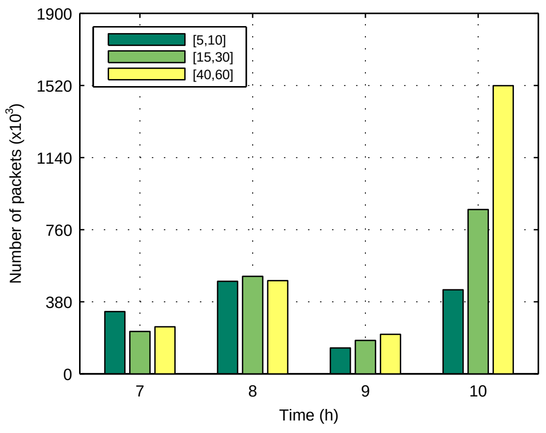


(b) - February 2015

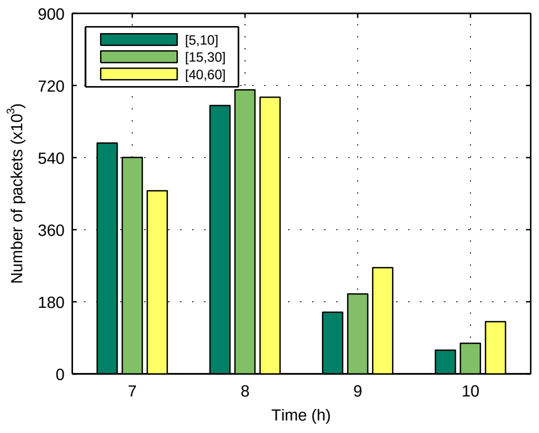
Figure 6.68: HelixEmulator evaluation impact of the advertisement packets periodicity during the rush hour period - Progress rate

Periodicity	**October 2014**		**February 2015**	
	Avg [h]	**C.I. (95%) [h]**	**Avg [h]**	**C.I. (95%) [h]**
[5,10]	1.77	±0.14	1.54	±0.12
[15,30]	2.04	±0.15	1.69	±0.13
[40,60]	2.52	±0.16	1.99	±0.15

Table 6.22: MatlabEmulator evaluation impact of the advertisement packets periodicity during the rush hour period - E2E delay statistics



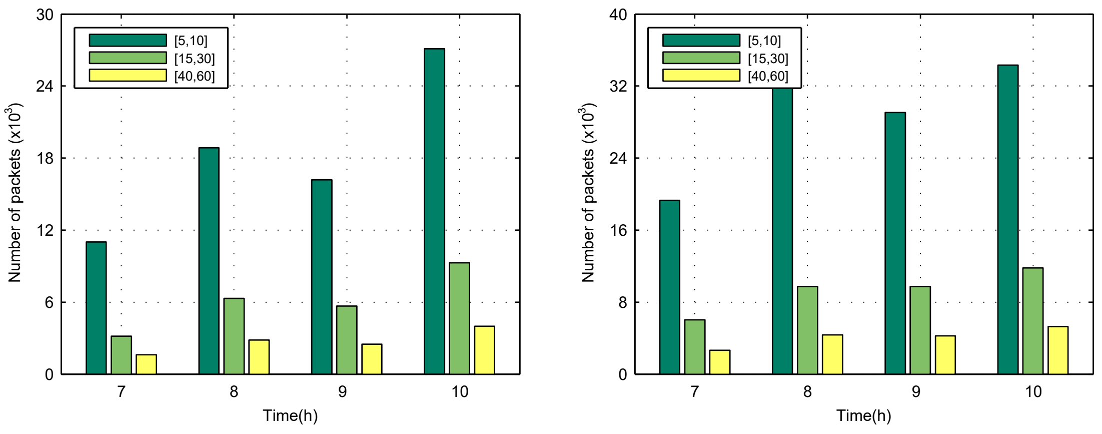
(a) - October 2014



(b) - February 2015

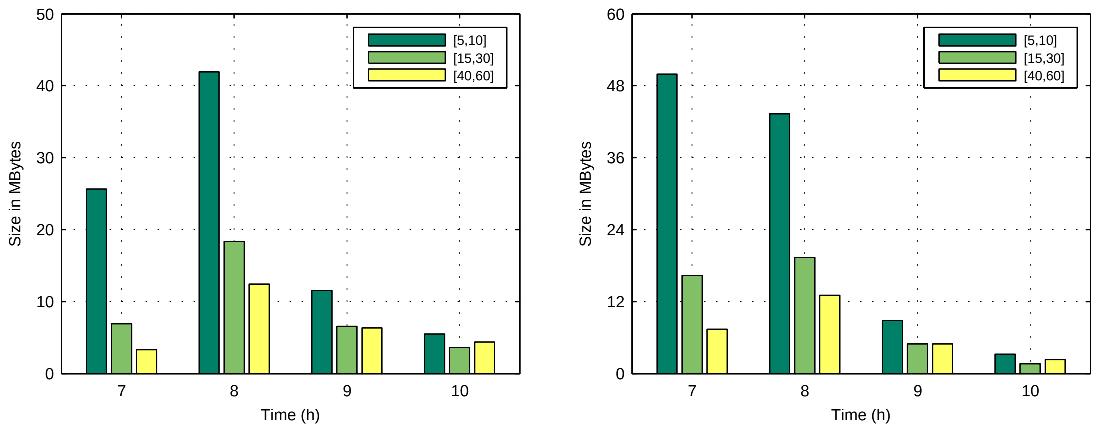
Figure 6.69: HelixEmulator evaluation impact of the advertisement packets periodicity during the rush hour period - Number of listened packets by the network (only OBUs) throughout the experiment

along with their total size. The results are as expected, since the number of transmitted advertisement packets increases when their dissemination is more frequent. This behavior remains until the end of the experiment. On the other hand, the overhead introduced by the advertisement packets (when their advertisement is more often) is higher in the beginning of the experiment, but tends to converge to the same value as the other strategies. This behavior is due to the fact that, when a node already has successfully downloaded the content, it sends a smaller packet compared to the one sent when it does not have all the content. Thus, since the delivery rate is higher when the advertisement broadcast is more frequent, the additional overhead decreases in the end of the experiment.



(a) - October 2014 (b) - February 2015

Figure 6.70: HelixEmulator evaluation impact of the advertisement packets periodicity during the rush hour period - Number of transmitted advertisement packets in network throughout the experiment



(a) - October 2014 (b) - February 2015

Figure 6.71: HelixEmulator evaluation impact of the advertisement packets periodicity during the rush hour period - Size of transmitted advertisement packets in network throughout the experiment

Table 6.23 summarizes the performance of the machine that is running the HelixEmulator. Thus, all metrics present worse values when the advertisement packets periodicity increases. This fact is due to the higher number of messages (data and control) exchanged in the

HelixEmulator when the delivery process is slower (see Figure 6.69).

	Periodicity	**CPU [%]**		**Load**		**Memory [%]**	
		Avg	Std	Avg	Std	Avg	Std
Oct 2014	[5,10]	9.1	11.9	1.3	1.8	23.6	5.6
	[15,30]	16.2	15.5	2.5	2.8	42.4	2.5
	[40,60]	25.8	20.9	5.2	3.4	44.9	1.4
Feb 2015	[5,10]	11.2	12.0	2.0	2.4	28.1	1.9
	[15,30]	9.6	10.0	1.5	2.3	43.7	1.4
	[40,60]	13.8	21.3	3.0	4.4	55.4	6.5

Table 6.23: HelixEmulator evaluation impact of the advertisement packets periodicity during the rush hour period - Computational performance statistics

Impact of the Time the Information is Valid

Another important parameter used by both LRBF and LRGF strategies is the valid time of an information in the node internal structures. For example, in the LRBF strategy, when a node updates the internal structure responsible for storing and organizing information about the vicinity storage content, this information only stays there for a certain period of (valid) time. Thus, it is important to evaluate what is the impact of that time in the dissemination process.

Three different valid times are tested: (i) 15 seconds, (ii) 22 seconds and (iii) 30 seconds. The range of these times are limited to the minimum and maximum default values of the refreshment procedure periodicity (15 and 30 seconds).

As can be observed in Figure 6.72 this parameter has a small effect in terms of delivery rate. Analyzing the delivery delay metric (illustrated in Figure 6.73, Figure 6.74, and Table 6.24) the conclusion is the same, wherein the different approaches present the same results. Regarding the network congestion illustrated in Figure 6.75, there are no major differences among the different approaches unless in the last hour of the October experiment. In this period, when a lower valid time is used, the number of listened packets is higher than in the other approaches. In terms of the network overhead and HelixEmulator performance, all experiments present similar values which lead to a conclusion that this parameter has a lower impact on the content distribution strategy. The previous statement is supported by Figure 6.76, Figure 6.77, and Table 6.25.

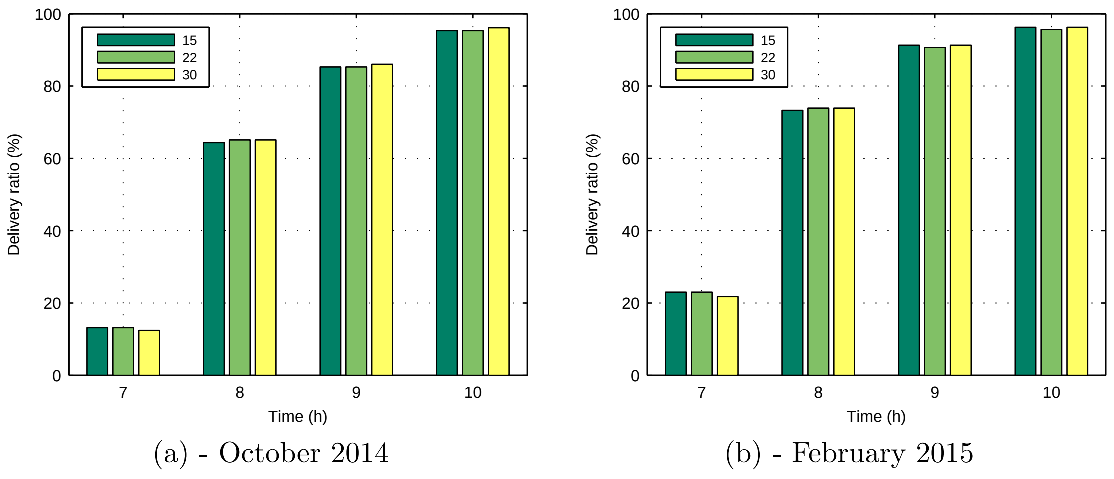
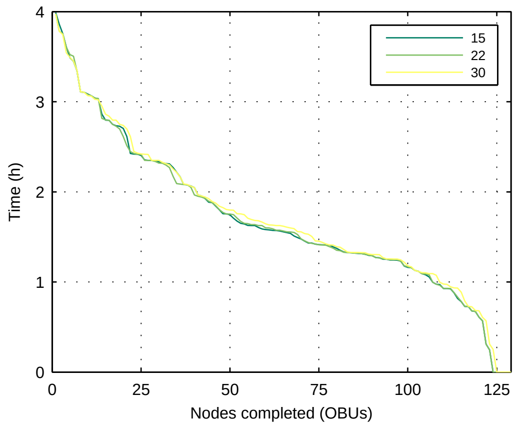
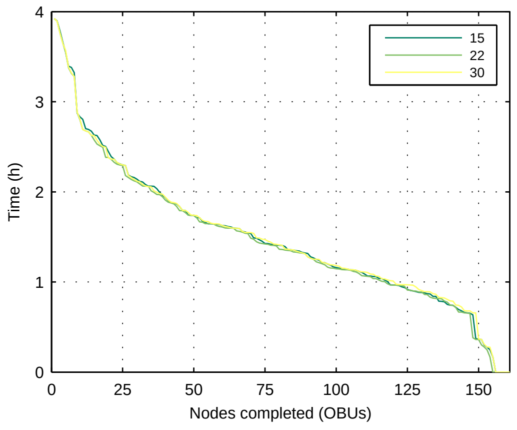


Figure 6.72: HelixEmulator evaluation impact of the valid information time during the rush hour period - Percentage of nodes with complete file per hour - Delivery rate

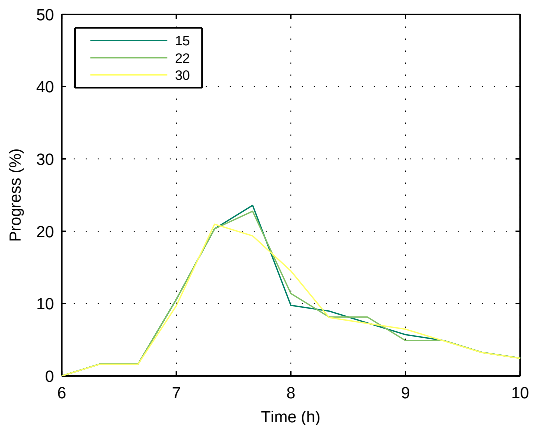


(a) - October 2014

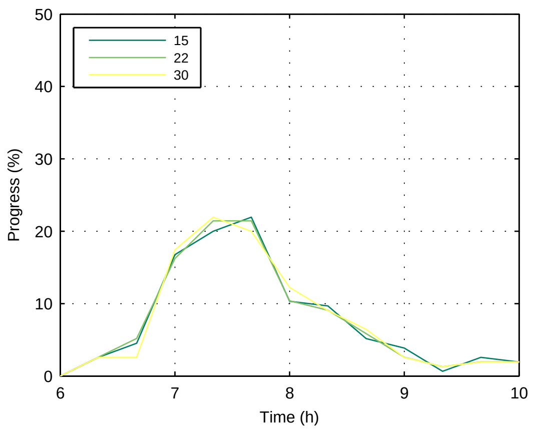


(b) - February 2015

Figure 6.73: HelixEmulator evaluation impact of the valid information time during the rush hour period - Time to receive the complete file per node - E2E delay



(a) - October 2014

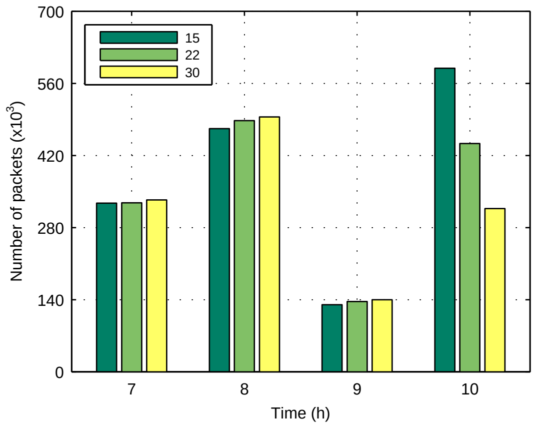


(b) - February 2015

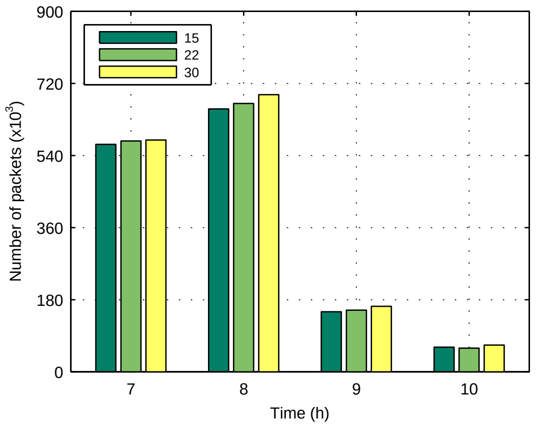
Figure 6.74: HelixEmulator evaluation impact of the valid information time during the rush hour period - Progress rate

Valid Time	**October 2014**		**February 2015**	
	Avg [h]	**C.I. (95%) [h]**	**Avg [h]**	**C.I. (95%) [h]**
15s	1.78	±0.14	1.75	±0.13
22s	1.77	±0.14	1.76	±0.14
30s	1.79	±0.13	1.75	±0.14

Table 6.24: MatlabEmulator evaluation impact of the valid information time during the rush hour period - E2E delay statistics

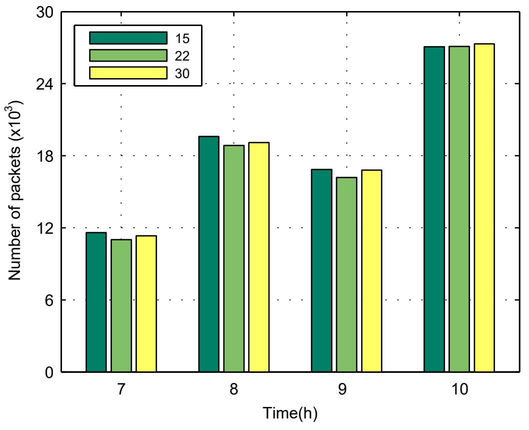


(a) - October 2014

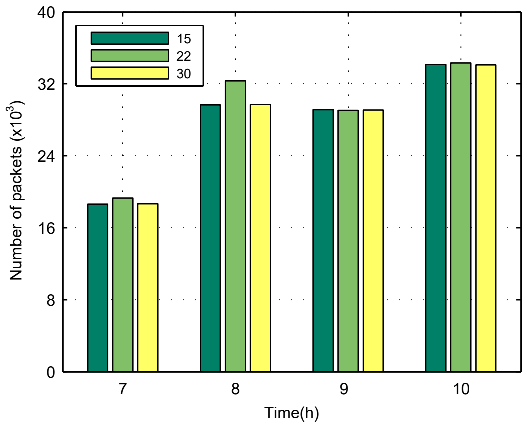


(b) - February 2015

Figure 6.75: HelixEmulator evaluation impact of the valid information time during the rush hour period - Number of listened packets by the network (only OBUs) throughout the experiment



(a) - October 2014

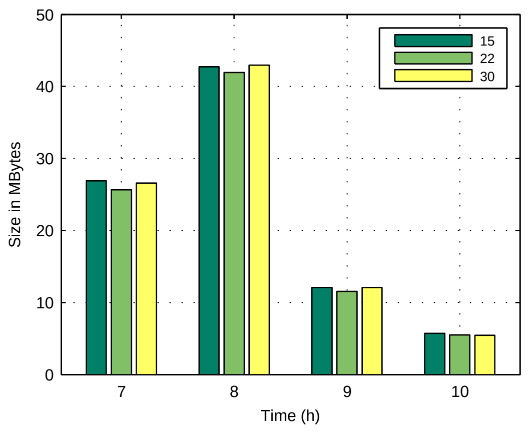


(b) - February 2015

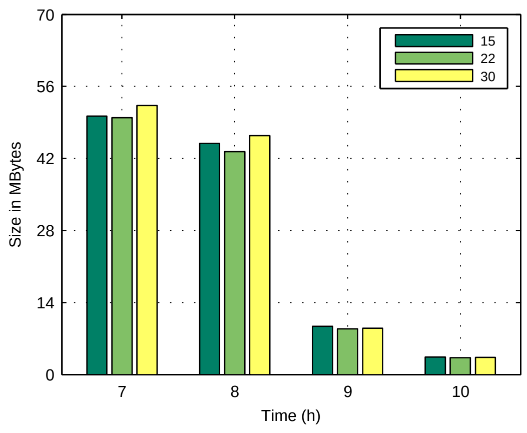
Figure 6.76: HelixEmulator evaluation impact of the valid information time during the rush hour period - Number of transmitted advertisement packets in network throughout the experiment

	Valid Time	**CPU [%]**		**Load**		**Memory [%]**	
		Avg	Std	Avg	Std	Avg	Std
Oct 2014	15s	14.7	3.4	1.8	2.1	43.4	1.2
	22s	14.2	12.0	2.0	2.4	28.1	1.9
	30s	14.0	3.1	2.0	2.5	43.5	1.2
Feb 2015	15s	9.1	11.8	1.3	1.8	26.0	2.7
	22s	9.1	11.9	1.3	1.8	23.6	5.6
	30s	9	11.4	1.4	1.9	26.9	2.7

Table 6.25: HelixEmulator evaluation impact of the valid information time during the rush hour period - Computational performance statistics



(a) - October 2014



(b) - February 2015

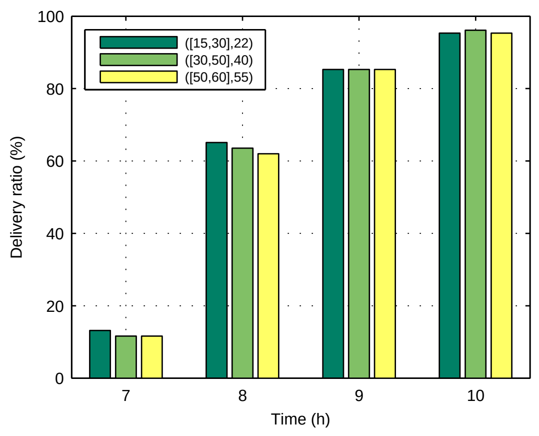
Figure 6.77: HelixEmulator evaluation impact of the valid information time during the rush hour period - Size of transmitted advertisement packets in network throughout the experiment

Impact of the Refreshment Periodicity

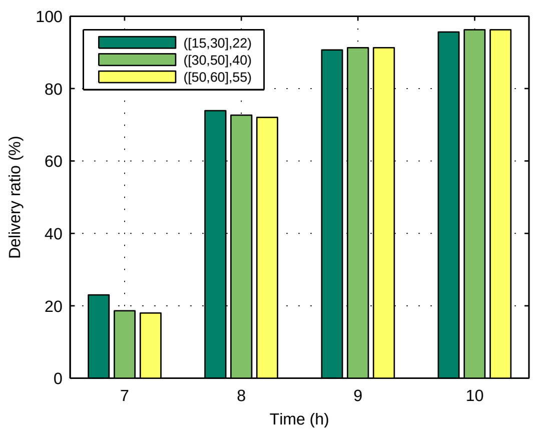
Finally, the refreshment periodicity to the internal structures is also tested. This is a very important parameter in the LRBF and LRGF strategies, since it controls the periodicity of the refreshment process which is responsible for validating if the information presented in such structures remain valid. For example, in the LRBF strategy, when a node updates the internal structures responsible for storing and organizing information about vicinity storage content, this information can only stay there for a certain period of (valid) time. The refreshment process is responsible to check the valid time of this type of information on a periodical basis. Thus, it is important to evaluate what is the impact of that time in the dissemination process.

Three different ranges of refreshment are tested: (i) from 15 to 30 seconds with a valid time of 22 seconds, (ii) from 30 to 50 seconds with a valid time of 40 seconds, and (iii) from 50 to 60 seconds with a valid time of 55 seconds. For each one of them the valid time is set in the middle.

As illustrated by Figure 6.78 this parameter has a small effect on the delivery rate. Regarding the delivery delay metric (illustrated in Figure 6.79, Figure 6.80, and Table 6.26), the same conclusions are valid, wherein the different approaches present the same results. Taking into account the network congestion illustrated in Figure 6.81, a smooth trend is observed. When the refreshment periodicity is high, the network congestion increases (more packets are transmitted). This behavior is due to a higher period of wrong information in the internal structures. If the refreshment is slower, a node thinks that the information in its internal structures is still good and continues the sending of data packets even if its vicinity already has the content (node does not have updated its structures yet). In terms of the network overhead and HelixEmulator performance, the experiments converge to similar values whereby the lower impact of this parameter in the content distribution strategy can be concluded. The previous statement is supported by Figure 6.82, Figure 6.83, and Table 6.27.

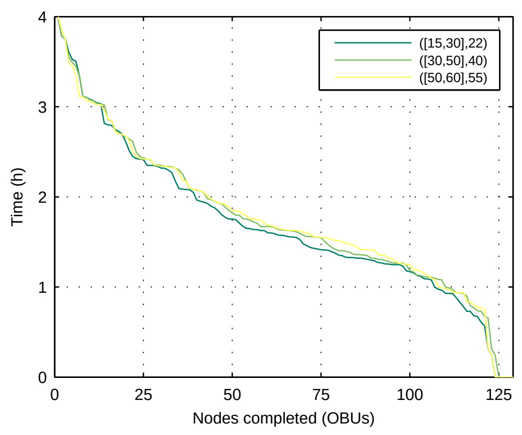


(a) - October 2014

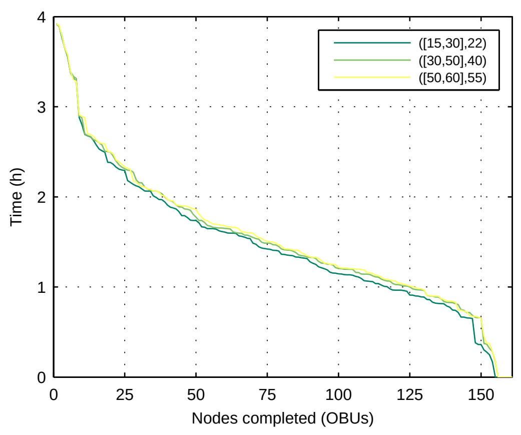


(b) - February 2015

Figure 6.78: HelixEmulator evaluation impact of the refreshment periodicity during the rush hour period - Percentage of nodes with complete file per hour - Delivery rate

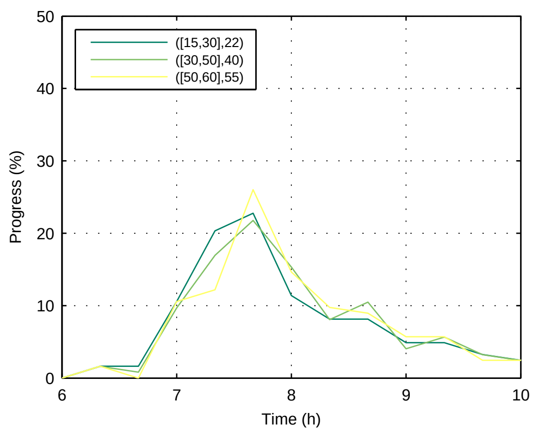


(a) - October 2014

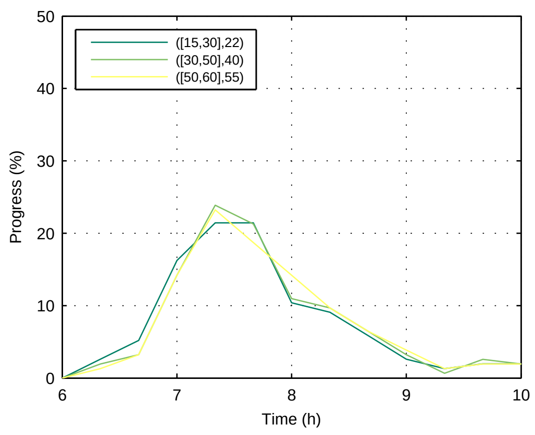


(b) - February 2015

Figure 6.79: HelixEmulator evaluation impact of the refreshment periodicity during the rush hour period - Time to receive the complete file per node - E2E delay



(a) - October 2014

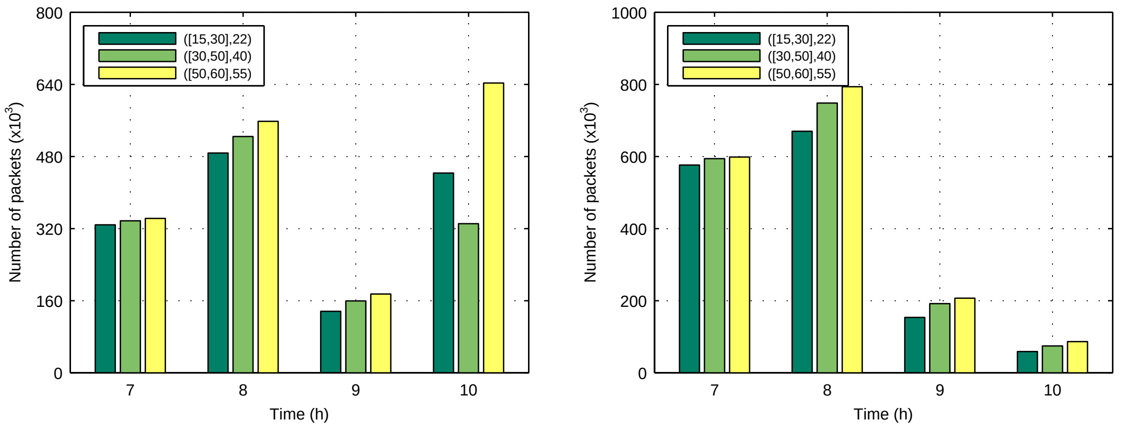


(b) - February 2015

Figure 6.80: HelixEmulator evaluation impact of the refreshment periodicity during the rush hour period - Progress rate

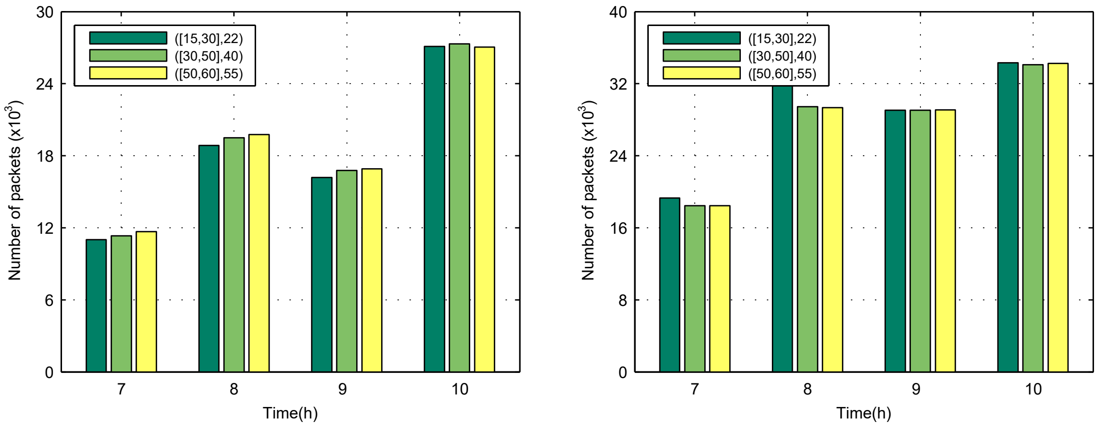
Periodicity	**October 2014**		**February 2015**	
	Avg [h]	**C.I. (95%) [h]**	**Avg [h]**	**C.I. (95%) [h]**
[15,30]	1.77	±0.14	1.54	±0.12
[30,50]	1.82	±0.14	1.59	±0.11
[50,60]	1.84	±0.13	1.61	±0.11

Table 6.26: MatlabEmulator evaluation impact of the refreshment periodicity during the rush hour period - E2E delay statistics



(a) - October 2014 (b) - February 2015

Figure 6.81: HelixEmulator evaluation impact of the refreshment periodicity during the rush hour period - Number of listened packets by the network (only OBUs) throughout the experiment

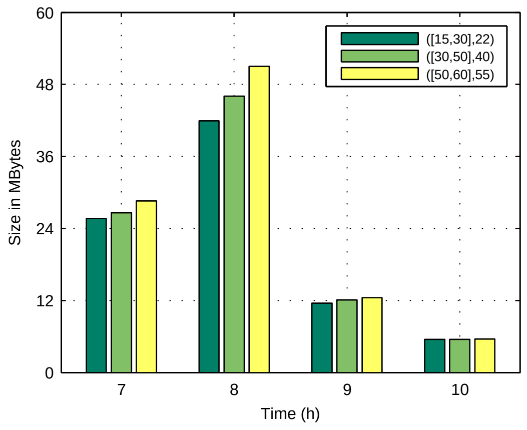


(a) - October 2014 (b) - February 2015

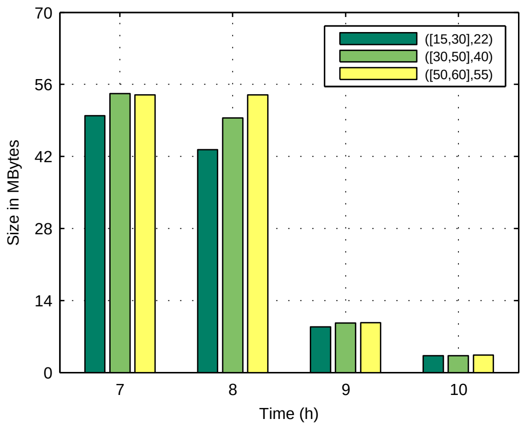
Figure 6.82: HelixEmulator evaluation impact of the refreshment periodicity during the rush hour period - Number of transmitted advertisement packets in network throughout the experiment

	Periodicity	**CPU [%]**		**Load**		**Memory [%]**	
		Avg	Std	Avg	Std	Avg	Std
Oct 2014	[15,30]	14.2	12.0	2.0	2.4	28.1	1.9
	[30,50]	14.3	13.2	1.9	2.2	44.0	1.3
	[50,60]	15.6	13.7	2.0	2.1	44.2	1.3
Feb 2015	[15,30]	9.1	11.9	1.3	1.8	23.6	5.6
	[30,50]	9.4	12.1	1.3	1.8	46.2	1.8
	[50,60]	9.7	12.3	1.3	1.8	46.9	1.9

Table 6.27: HelixEmulator evaluation impact of the refreshment periodicity during the rush hour period - Computational performance statistics



(a) - October 2014



(b) - February 2015

Figure 6.83: HelixEmulator evaluation impact of the refreshment periodicity during the rush hour period - Size of transmitted advertisement packets in network throughout the experiment

6.9 Laboratory Evaluation

Since until this moment all the evaluation procedures were performed in an emulation environment, a real testbed was developed in order to evaluate the behavior of the best content distribution strategy - LRBF. As described in section 6.4, this testbed is composed by a small set of nodes and aims to prove that this strategy correctly runs on a real network.

6.9.1 Considerations

Before the analysis of the results it is important to list a set of considerations and definitions taken into consideration in the deployment of this scenario.

Before starting to run the Helix software, the IEEE 802.11p channel is configured in order to provide inter-communication among nodes. The characteristics of this channel are described in Table 6.28. As claimed by [151] the NetRider boards use an application called Universal WAVE Management Entity (UWME) that uses a multi-channel approach for transmission. Two types of channels are specified in the WAVE standard: CCH and SCH. In this experiment the CCH is continuously operating and a request is sent to access the SCH when the Helix needs to perform a communication task.

Number	176
Provider Service ID	80-10
Mode	Continuous
Tx Power	23 dBm
Max Bit Rate	27 Mbps

Table 6.28: IEEE 802.11p channel characteristics used in laboratory tests

The Helix configuration file was configured as described in Table 6.29. This table only represents the most relevant parameters like the number of the socket port, the storage capacity, and the wireless interface used. In the experiment the communication was always performed using the IEEE 802.11p interface since the nodes were only of two types (OBUs and RSU) who tend to use this kind of technology.

Socket Port	4556
Storage Capacity	20 MB
11p Interface	wlan1

Table 6.29: Helix configuration in OBUs and RSU

Regarding the content advertisement strategy configuration, a set of internal macros were defined as described in Table 6.30. Thus, an advertisement period of 5 seconds was configured, the information of internal structures was valid for a period of 30 seconds and these were updated with a periodicity of 30 seconds. Moreover, the Logging module was configured to record logs with a periodicity of 10 seconds.

Contrarily to the emulation, in the laboratory experiment the bandwidth limitation of 1 Mbps was not imposed, but the transmission remains opportunistic whereby a node only sends packets when there is at least one neighbor in its vicinity. However, the transmission of information has a limitation imposed by the Routing module of Helix. To decrease the CPU usage, a sender node sleeps a specific time between sent packets. The value for that sleep was

Advertisement Periodicity (`CONT_ADV_PERIOD`)	5 s
Refreshment Periodicity (`REFRESH_PERIOD`)	30 s
Information Valid Time (`ELEMENT_VALID_TIME`)	30 s
Logging Periodicity (`TIME_STEP_LOG`)	10 s

Table 6.30: LRBF configuration for laboratory evaluation

defined as 100 ms whereby a node can send a maximum of 10 packets per second which gives a maximum transmission rate of 2.6 Mbps (considering packets of 32 KB).

Regarding a correct time synchronization among nodes, all of them were running NTP in background in a client-server model. At the beginning of the experiment the clock of the RSU number 172 is forced and the board starts running NTP as a server. All the other nodes of the network run the NTP as clients and define the server with the IP of IEEE 802.11p interface of the RSU. Using this approach it is guaranteed that all nodes are synchronized before the start of the experiment.

6.9.2 Results

In this section the most relevant metrics registered during the laboratory evaluation are presented and discussed. These outputs result from a set of 9 repetitions of the experiment described in subsection 6.4.2. The file under dissemination is divided in 320 pieces of 32 KB each, which represents a total size of approximately 10 MB. For the evaluation of the results, a 95% confidence interval was considered.

Figure 6.84 presents the percentage of nodes which received the complete content along the experiment. It is clear that in all the iterations of the experiment all the nodes received the file and took a maximum time of 200 seconds to do so.

In Figure 6.84 there are two periods where the evolution of the delivery rate has a more abrupt slope. The first one is between the seconds 120 and 150, and it results from the delivery of the file to the mobile OBU since in this period it is stopped in a region covered by the RSU. Thus, in the majority of the iteration at least two nodes had the content (mobile OBU and fixed OBU number 132). The second period is defined in the range between 150 and 210 seconds. In this period the other two fixed OBUs downloaded the file since the mobile node already had the complete file and was able to deliver it during its traveling.

Figure 6.85 confirms the previous analysis, since it describes the percentage of file downloaded by each node during the experiment. Thus, the first node to receive the file is the fixed OBU number 132 since it has direct access to the source of information. Due to the periods of direct contact with the RSU, the mobile node is the second node that downloads the file. After the 150 seconds, the other two fixed OBUs download the file through the mobile node. The mean E2E delay regarding the delivery of the content to all the receiver nodes is shown in Figure 6.86. Thus, this is another evidence of the previously described network behavior in terms of the time to deliver the complete content throughout the network.

The number of listened packets is approximately uniform across all the nodes as Figure 6.87 illustrates. The OBU 132 has the lower number of listened packets, since it only listens to packets from the RSU and OBU number 107, and the RSU only broadcasts information until the mobile node and the OBU number 132 do not have all the content. On the other hand, OBU number 109 has the highest number of listened packets, since it listens information from OBUs 107, 237 and sometimes (with some reflections) from 132.

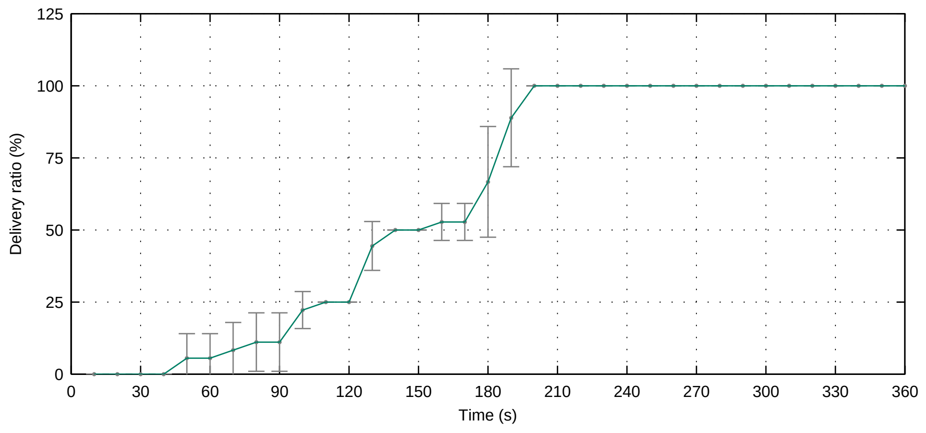


Figure 6.84: Laboratory evaluation - Percentage of nodes with complete file per T_s - Delivery rate

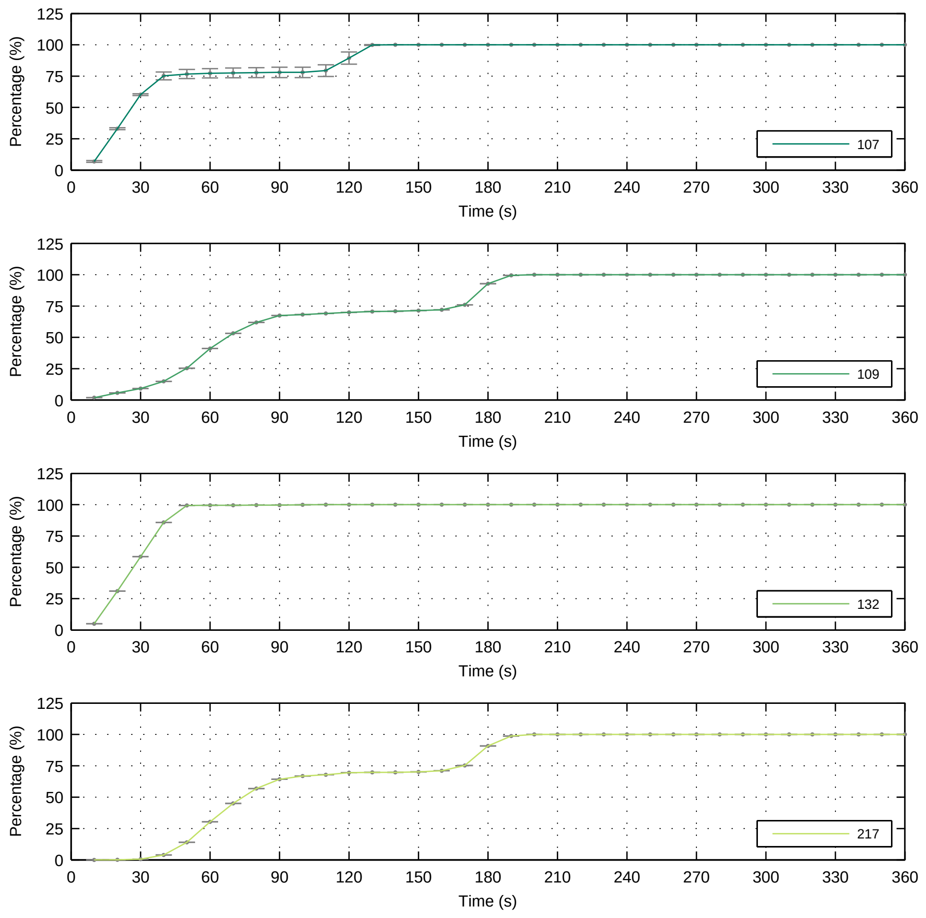


Figure 6.85: Laboratory evaluation - Percentage of file distributed per node throughout the experiment

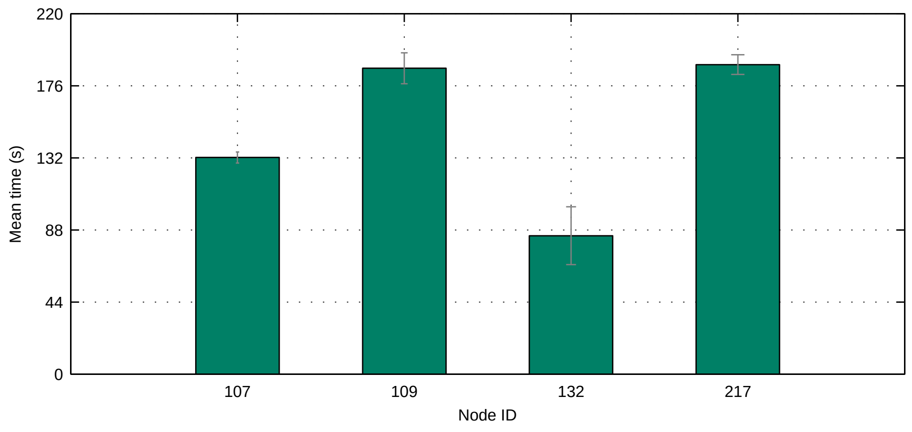


Figure 6.86: Laboratory evaluation - Mean time to receive the complete file per node - E2E delay

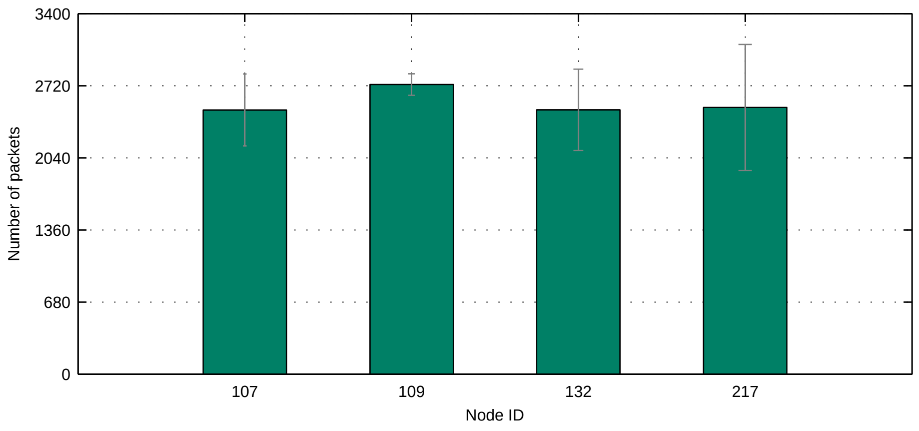


Figure 6.87: Laboratory evaluation - Total number of listened packets per node throughout the experiment

---

Figure 6.88 shows the total number of packets transmitted by the nodes along the experiment. The implemented strategy was based on an assumption that, when a node has an RSU as neighbor, the probability that its vicinity also has the same RSU as neighbor is high, whereby this node does not send any packet in this situation. Thus, the fixed OBU with direct contact with the source of information does not send any packets during this experiment.

On the other hand, the RSU sends information until all the nodes within its vicinity have the content, whereby it has a significant number of packets transmitted to the network. Moreover, the number of transmitted packets from the RSU is also limited by the fact that it only sends data packets while its vicinity does not have all the content (this decision is imposed by a set of advertisement messages). Due to a high end-to-end delay of delivery in the fixed OBUs further away from the source of the information, they send more packets since they are transmitting during a longer time period.

As shown in Figure 6.89, it is clear that the number of transmitted advertisement packets is similar among all network nodes. This was expected since the sending of advertisements is periodic and triggered by the reception of an advertisement packet with information about

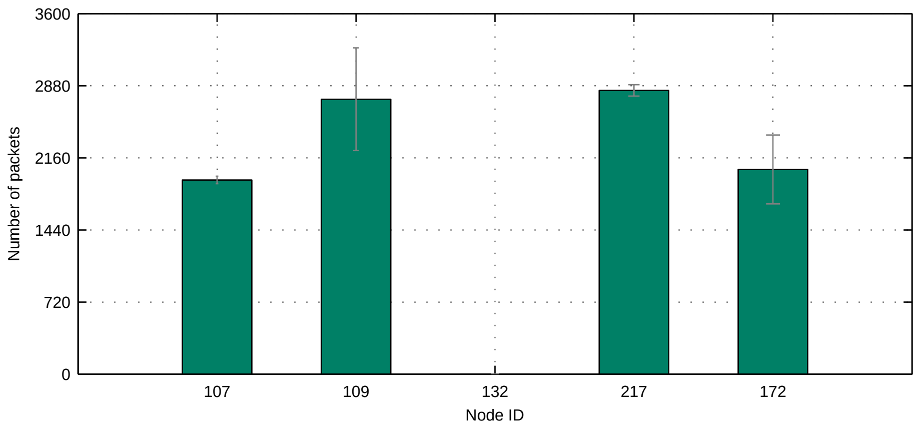


Figure 6.88: Laboratory evaluation - Total number of transmitted packets per node throughout the experiment

a new content to be downloaded. Thus, the fixed OBU number 217 sends less advertisement packets, since it is the last node to received the information about a new file to be downloaded.

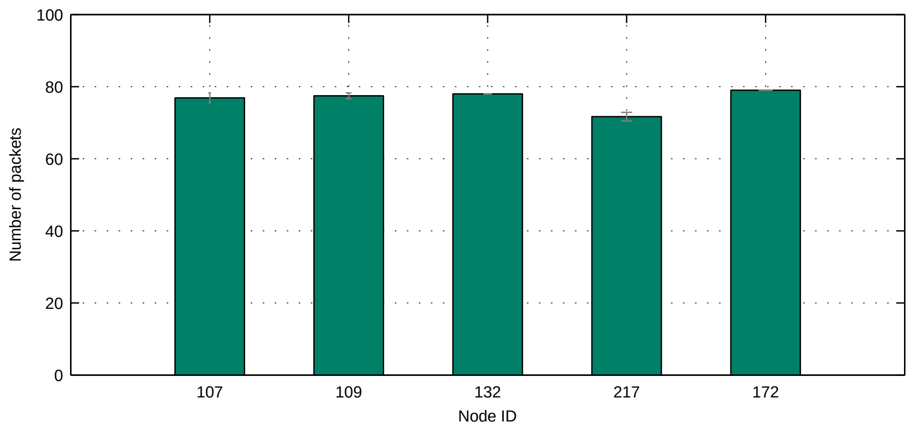


Figure 6.89: Laboratory evaluation - Total number of transmitted advertisement packets per node throughout the experiment

Figure 6.90 illustrates the mean size of all the advertisement packets transmitted during a sample period throughout the experiment. As it is clear in the figure, this metric presents higher values in the period between 60 and 90 seconds, since this is the period where the majority of the nodes do not have the content and broadcast advertisement packets signaling its storage content. Once all the nodes have the content, the size of the advertisement packets is the same across the network and is lower than in the beginning, since it only advertises the profile of the content under dissemination.

Figure 6.91 represents the total size of the transmitted advertisement packets per node throughout the experiment, and it is relevant to analyze the impact of the advertisement packets in the network overhead. Table 6.31 combines this information with the number of data packets transmitted by each node (considering each one of them has a 32 KB packet).

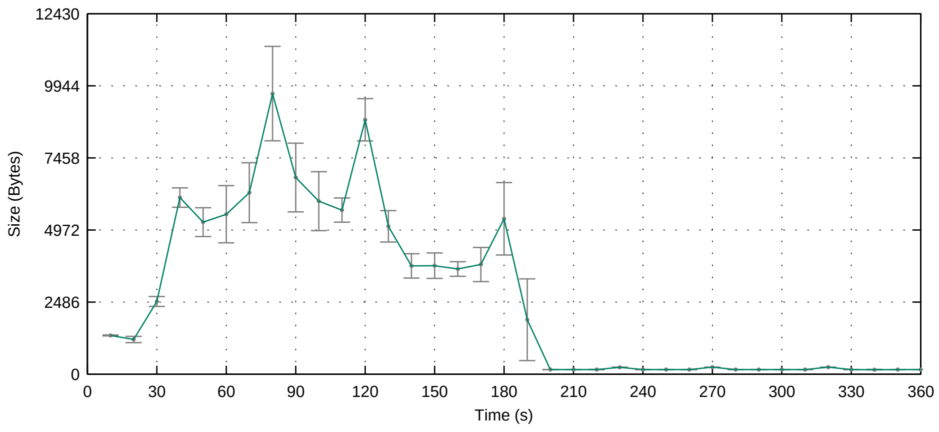


Figure 6.90: Laboratory evaluation - Mean size of all transmitted advertisement packets in the network throughout the experiment

---

Thus, it is possible to conclude that the advertisement adds a small overhead to the network since the great majority of the disseminated packets are data packets.

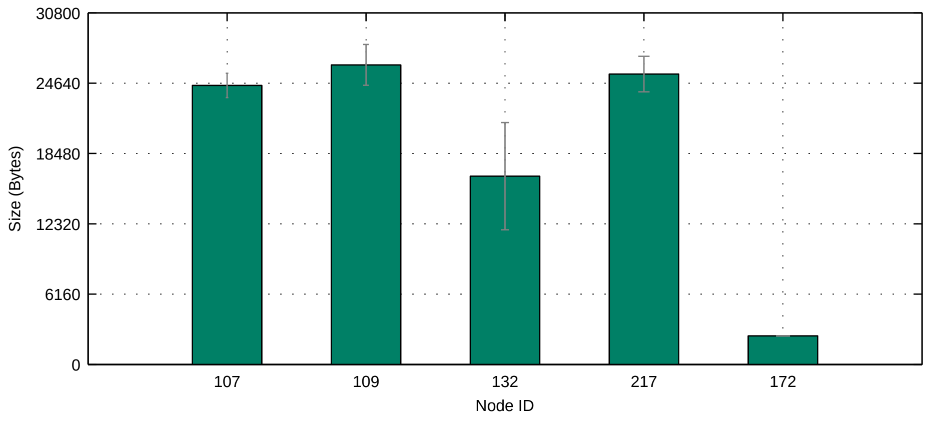


Figure 6.91: Laboratory evaluation - Total size of transmitted advertisement packets per node throughout the experiment

Node ID	**Transmitted Packets (Data+ADVs)**	**Data Packets**		**ADVs Packets**	
	[KB]	[KB]	[%]	[KB]	[%]
107	62075.96	62051.52	99.96	24.44	0.04
109	87880.32	87854.08	99.97	26.24	0.03
132	16.49	0	0	16.49	100
217	90695.52	90670.08	99.97	25.44	0.03

Table 6.31: Laboratory evaluation - weight of the advertisement packets in the dissemination process

The last three figures (Figure 6.92, Figure 6.93 and Figure 6.94) are related to the network resource usage. Three metrics are analyzed: CPU usage, load, and used memory RAM. All the parameters present normal values which proves that the strategy implemented does not overload the board resources.

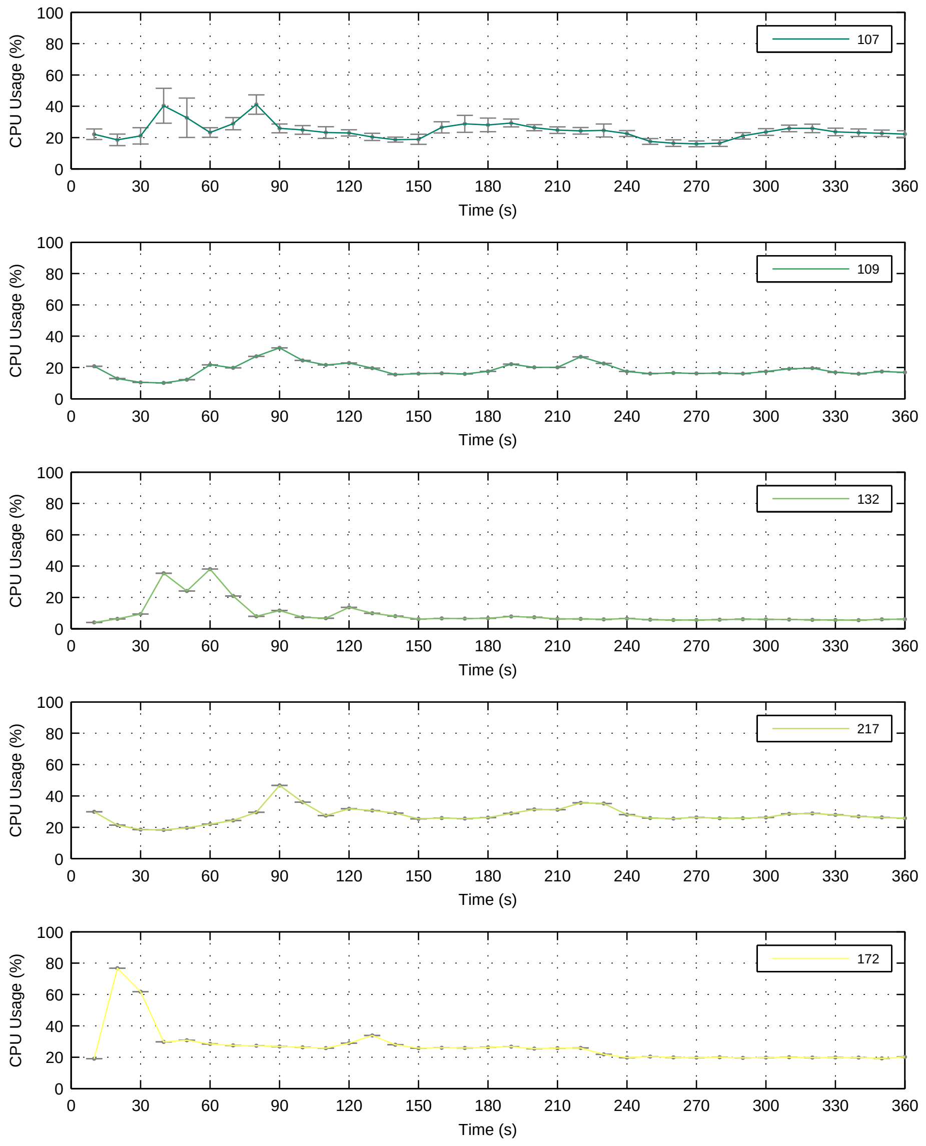


Figure 6.92: Laboratory evaluation - CPU usage per node throughout the experiment

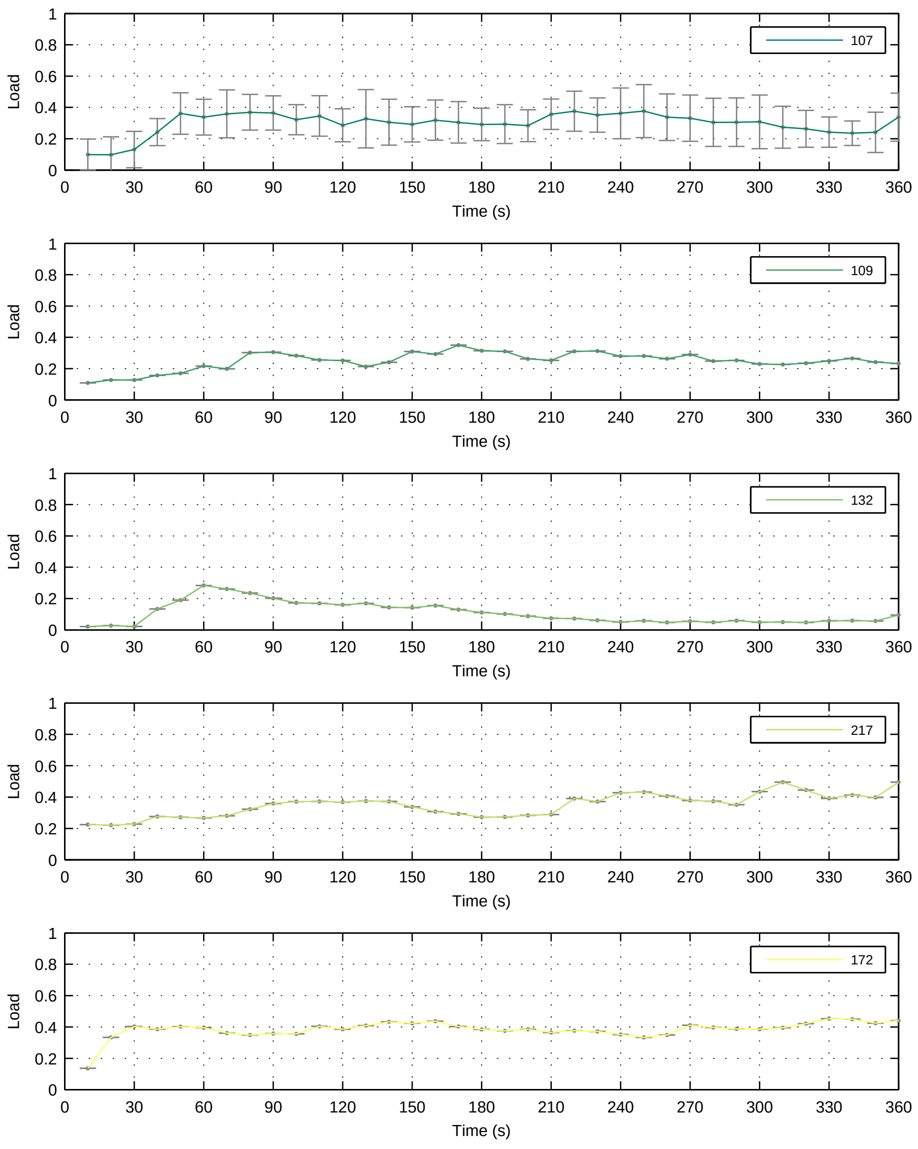


Figure 6.93: Laboratory evaluation - Load per node throughout the experiment

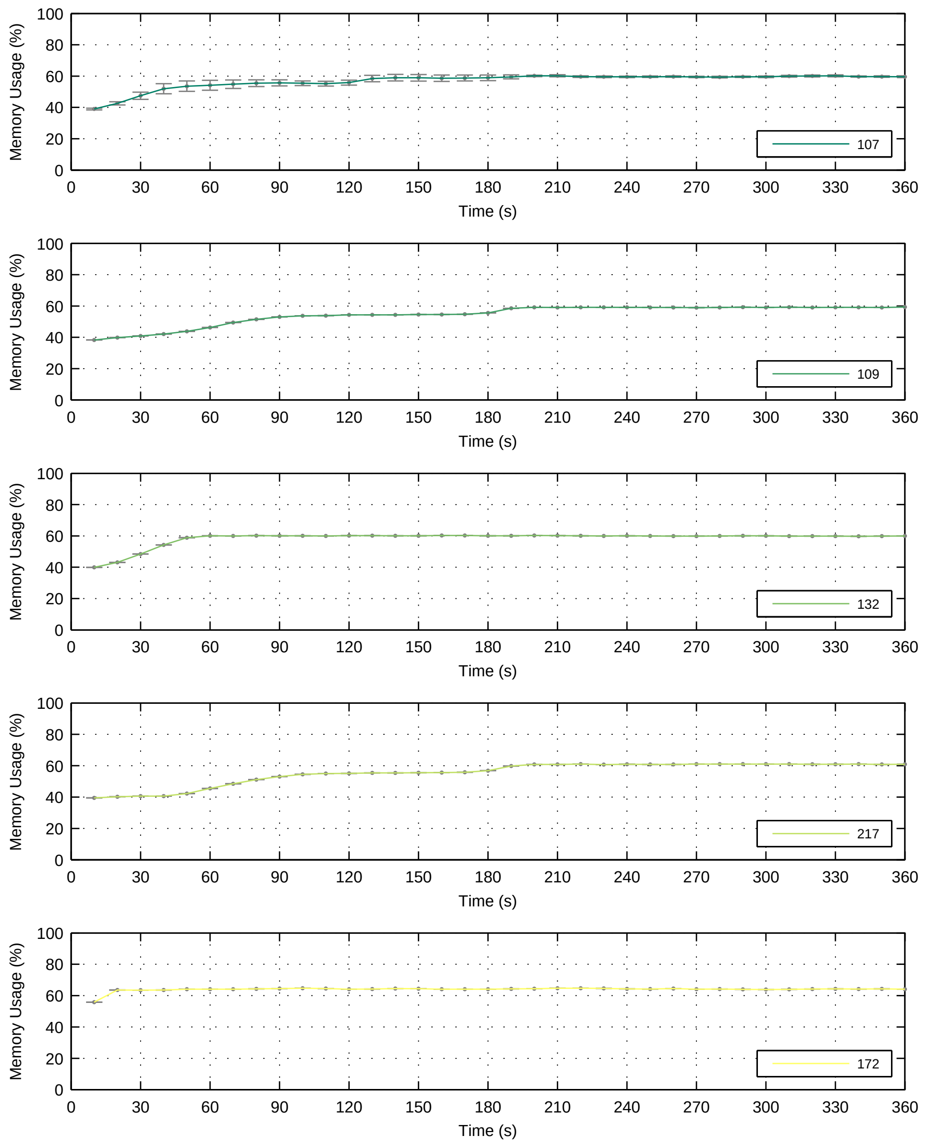


Figure 6.94: Laboratory evaluation - Memory usage per node throughout the experiment

6.10 Chapter Considerations

This Chapter focused on the evaluation of the content distribution schemes proposed in Chapter 4, along with an initial study of the vehicular network under the scope of this work. A large set of platforms was used to evaluate those strategies and perform this study. The main considerations of this Chapter are described as follows.

Initial Study of the Network

In the beginning of this Chapter the scenarios under evaluation were presented and described. In this description, several aspects related with the network infrastructure (number and location of the RSUs) and network vehicles (number of OBUs) were described. This initial description allowed for a better understanding of the potential of the network in the different evaluation periods.

Once the potential of the network's infrastructure was explored, a detailed statistical analysis was performed in order to trace a profile of the two datasets used as input mobility models in the used emulation platforms. Three metrics were evaluated: active time of OBUs, number of contacts of OBUs and the mobility trends of OBUs. Through this evaluation it was possible to trace common profiles between the different datasets. Thus, the vehicles (OBUs) tend to be more active in the rush-hour periods (early morning and afternoon), along with the periods that they are parked (midday and evening). During the parking periods the most common type of neighbors are STAs; during the rest of the day these are OBUs and RSUs, since the vehicles travel along the city center where the fixed-infrastructure is more present.

These considerations were very relevant in order to define the evaluation periods. Thus, a set of experiments where defined, taking into account the time of day and the geographical area.

Content Distribution Schemes Evauation in Emulation Platforms

Two emulators were created with the main goal to evaluate content distribution schemes using delay tolerant mechanisms. Thus, these two platforms (MatlabEmulator and HelixEmulator) were used to evaluate the proposed schemes.

Even though these emulators are different, it was possible to obtain similar results on both. Thus, the evaluation procedure concludes that the LRBF strategy presented a higher delivery rate and lower delay. The second best strategy was the LRGF, which most of the time presented a final delivery rate equal to the LRBF, but with a higher delay. The other two strategies, Random and LNHF, are clearly weaker than the previous two. The LRBF strategy presented a drawback when compared to the LRGF strategy, since it introduced a higher network overhead due to the larger advertisement packets.

Several parameters related with the content distribution strategies were also evaluated. After this evaluation, the following parameters can be identified as critical in the configuration of the proposed content distribution strategies. Evidently, the size of the content under dissemination is a major factor since the bigger it is, the lower the delivery will be and the higher the delay will be. The periodicity of the advertisement packet also has a major impact on content dissemination, since if these packets are sent more often, the probability of an accurate forwarding decision (it forwards the most lacking packet) is higher than with a lower frequency of advertisement broadcasting.

The strategies to disseminate information presented in Chapter 4 were also evaluated. The proposed strategy to optimize the delivery was only effective in the parking period where the density of nodes is higher and enables a higher delivery rate. On the other hand, the strategy proposed to minimize the network congestion produced the expected results, decreasing network congestion along with enhancing the delivery rate. The proposed hybrid approach also enhances the applied content distribution strategy (LRBF). Thus, the proposed strategies to disseminate information proved their advantages and should be considered in future experiments.

LRBF Strategy Evauation in a Laboratorial Platform

Finally, the strategy with the best overall performance was evaluated in a real scenario. The selected strategy was the LRBF and the main goal of this experiment was to confirm the correct behavior of this strategy in a real scenario. The experiment was performed in the laboratory using the same OBUs that are used in the Oporto vehicular network. The evaluation confirmed the correct development and deployment of the proposed strategy, and also confirms that the content distribution service is ready to be deployed and evaluated in the Oporto city vehicular network.

Chapter 7

Conclusions and Future Work

7.1 Conclusions

The main goal of this dissertation was the study, design, implementation and evaluation of several strategies for content distribution in a vehicular network. As such, four strategies were implemented: Random, LNHF, LRBF, and LRGF. Additionally, this dissertation also comprised the study of several techniques to minimize network congestion and maximize content delivery.

The following strategies were defined based on existing literature. The Random strategy was proposed with the goal of understanding the efficiency of a purely random decision strategy and that does not add any overhead to the network in order to make routing decisions. The second strategy (LNHF) focuses on a metric that introduces a low overhead to the network, but that also allows the sender node to make a more intelligent routing decision. Therefore, it was defined that, for the RSUss this metric would be the number of transmissions, and for the OBUss it would be the number of nodes through which a certain packet has already passed. This approach leads to the RSUs having a propensity to send the least disseminated packet, since it is the least emitted packet, and the probability that it has traveled through a large number of nodes is lower. The thought process for the OBUs is similar, since a packet that has a larger number of hops has probably been dowloaded less times. The last two strategies were based on increasing the status of nodes, but improving the delivery. The first strategy focuses on the implementation of a P2P dissemination in which the sender node knows which is the most lacking packet in its vicinity and proceeds to send it. This knowledge is obtained through the dissemination of control messages from each node, which adds overhead to the network. The last implemented strategy uses the concept of network coding to disseminate content. This concept introduces redundancy to the network, increasing the possibility of various network elements receiving the same content. Given that the communication between network elements occurs through broadcast dissemination, there are many challenges to face. Therefore, several techniques were suggested in order to minimize these effects. These techniques focus on using network knowledge (number of neighbors) and packet-specific information (number of hops) in order to minimize network congestion and maximize delivery.

A number of challenges appeared during the development of this work, sometimes delaying implementation and evaluation of the proposed strategies. However, these challenges were overcome and all the four proposed strategies are considered stable. Several challenges were identified that related to the design and rapid tests of the dissemination strategies to be implemented on the real testbed's software. A MATLAB emulator was created in order to facilitate the process of strategy design thinking. This emulator recreates (in an extremely similar way) the target vehicular network – the Future Cities project's vehicular network – as well as the content distribution process for each of the aforementioned strategies. This initial analysis allowed for the identification of the strategies and techniques that displayed the most potential for real network implementation. Another challenge was related to the difficulty in testing the behavior of the different strategies (that were already implemented in the target vehicular network's software) on a scale that was large enough to produce valid results. As such, and in order to ensure the aforementioned premise, it was developed the strategies based on a network emulator – the HelixEmulator. An initial version of this emulator was concluded over the course of the development of this Dissertation. Even though this emulator performed all basic functionalities, it was necessary to introduce several modifications in order to implement the new features required to evaluate the proposed content distribution strategies. Therefore, the emulator was equipped with additional capabilities, such as packet sending in broadcast, a bandwidth control module, a log file generating module (for subsequent statistical analysis), limitation in terms of emulating nodes, among others.

The HelixEmulator is responsible for launching a set of processes in equal number to the number of emulation nodes. Each one of these processes represents a network node that is running the Helix software. As such, this emulator also allows to directly generate Helix source code for the NetRider boards. Considering this, all of the necessary modifications needed for the implementation and integration of the four strategies were tested on the HelixEmulator. The routing module – responsible for the decision to send packets – was not sufficiently developed. As such, the core of this work focused on the development of the routing module in order to provide new features that would allow for the implementation of a set of content distribution strategies in delay-tolerant vehicular networks.

Several platforms were used to evaluate the different proposed content distribution strategies. The MatlabEmulator was used for the design and easy and quick testing of the several strategies. After the initial design, the implementation and integration of the routing strategies and techniques on the Helix software were done. In this evaluation process, several scenarios were assessed. The evaluated process used data that was collected during a 24-hour period in the FutureCities project vehicular network to emulate node mobility in an approximate way. Two scenarios were tested on the different platform and on the different time periods. The behavior of the proposed strategies was evaluated when the dissemination was focused on the city center and when the vehicles were concentrated on a single site (the parking lot of the public transportation company of Oporto city). Three separate time periods were tested in order to evaluate the impact of different mobility patterns of the vehicles on content distribution.

After analyzing the behavior of the proposed content distribution strategies in the multiple scenarios it was possible to draw several conclusions. There are two strategies that clearly set themselves apart from the others due to their superior performance: LRBF e LRGF. These strategies have delivery rates that are much higher than the ones from the Random and LNHF strategies, and lead to less network congestion. However, these strategies require additional control information to perform their routing decision, which increases the network overhead. Considering this aspect, the LRGF had the best results since it requires less information to be disseminated when compared to LRBF. In terms of delivery rate and delay of the aforementioned overall best strategies, the LRBF displayed higher delivery rates and less delay. The evaluation on the MatlabEmulator and HelixEmulator allowed for the conclusion that the strategy that provided higher assurance in terms of delivery and delay was the LRBF. Therefore, this was the strategy that was chosen to be implemented on a laboratory testbed composed of NetRider boards. The laboratory assessment proved the correct implementation of the LRBF strategy and that this strategy is suited for content distribution on a vehicular network. The scalability of the several strategies was also tested and evaluated during the experiments on the MatlabEmulator and the HelixEmulator.

With all the results gathered, it was then possible to draw some conclusions about the proposed content distribution strategies' performance. All summed up, it is safe to assume that good results were achieved with the developed implementation of the LRBF strategy given that the content dissemination is ensured with good delivery rate and delay time. This fact makes it possible to consider LRBF strategy as a safe choice for the content distribution in a vehicular environment.

7.2 Future Work

This Dissertation was the first one performed in our group regarding this study: development and implementation of a content distribution strategy. Thus, not all topics were analyzed and evaluated whereby several improvements and future topics of research and development should be addressed. Some of these topics can be summarized as follows:

- *Real network testing and evaluation*: even though all of the strategies were tested and evaluated, most of them were only tested in an emulation context. The LRBF strategy was tested on the boards that are used in the FutureCities network in laboratorial context. Therefore, in the future the other strategies should be tested on the real testbed in order to assess the behavior of the LRBF e LRGF in a real vehicular environment with a high number of nodes.

- *Improvement of HelixEmulator performance*: given the problems that were identified when the HelixEmulator attempts to emulate a high number of nodes in a highly congested network, the emulation process should be revised in order to assess the reason why CPU usage increases uncontrollably, eventually leading to the collapse of the machine that is running the emulator. Once the reason is identified, improvements should be made in order to mitigate this issue.

- *Design and integration of a codification module*: in the LRGF strategy the codification module was approximated by the introduction of redundant packets in the network. In a real context, this process should be implemented through a codification module that has not yet been developed.

- *Additional and improved network congestion control strategies*: in this work several strategies to optimize the delivery and minimize the network congestion were proposed. However, most of them relied on parameters related with highly variable metrics such as the number of hops and neighbors. Thus, in the future these parameters should be defined using fitness functions or machine learning algorithms in order to define the most satisfying value for them.

- *Strategies for content dissemination*: other strategies that limit the network overhead shall be identified and assessed, for example, considering DHTs (Dynamic Hash Tables) and content reconciliation.
- *Deployment of a coding module*: in this work the concept of network coding was discussed, concluding that this brings advantages for content dissemination. However, this work was only focused on the decision of which coded packets should be sent, assuming that the coding module was running and coded packets were available to send. Thus, in order to evaluate the LRGF more accurately, the introduction of a coding module that works in Helix software is highly recommended.
- *APIs for content distribution*: the content that was distributed through the course of this Dissertation was created directly in the source code. However, this process should be improved by creating APIs that are dedicated to the creation and sending of contents in order to facilitate the implementation of this service on a real network.
- *Interoperability of multiple services*: over the course of this Dissertation all of the tests were performed considering the exclusiveness of services on the network, i.e., only the content distribution service was activated. As such, broader tests that promote the coexistence of several services simultaneously should be performed.

Bibliography

[1] Veniam. (2015, August) An internet of moving things. [Online]. Available: https://veniam.com/

[2] V. Cerf, S. Burleigh, A. Hooke, L. Torgerson, R. Durst, K. Scott, K. Fall, and H. Weiss, "Delay-Tolerant Networking Architecture," RFC 4838 (Informational), Internet Engineering Task Force, Apr. 2007. [Online]. Available: http://www.ietf.org/rfc/rfc4838.txt

[3] G. W. Euler, "Intelligent vehicle/highway systems: Definitions and applications," *ITE journal*, vol. 60, no. 11, pp. 17–22, 1990.

[4] R. Uzcategui and G. Acosta-Marum, "Wave: a tutorial," *Communications Magazine, IEEE*, vol. 47, no. 5, pp. 126–133, 2009.

[5] "Ieee standard for information technology– local and metropolitan area networks– specific requirements– part 11: Wireless lan medium access control (mac) and physical layer (phy) specifications amendment 6: Wireless access in vehicular environments," *IEEE Std 802.11p-2010 (Amendment to IEEE Std 802.11-2007 as amended by IEEE Std 802.11k-2008, IEEE Std 802.11r-2008, IEEE Std 802.11y-2008, IEEE Std 802.11n-2009, and IEEE Std 802.11w-2009)*, pp. 1–51, July 2010.

[6] "Ieee standard for wireless access in vehicular environments (wave)–multi-channel operation," *IEEE Std 1609.4-2010 (Revision of IEEE Std 1609.4-2006)*, pp. 1–89, Feb 2011.

[7] Carnegie Melloon Portugal. (2015, July) Drive-in – distributed routing and infotainment through vehicular inter-networking. [Online]. Available: http://www.cmuportugal.org/tiercontent.aspx?id=1552

[8] FutureCities Project. (2015, July) Living lab : Vehicular ad-hoc networking. [Online]. Available: http://futurecities.up.pt/site/vehicular-ad-hoc-networking-testbed/

[9] SAFESPOT. (2015, July) Safespot. [Online]. Available: http://www.safespot-eu.org/

[10] NEC. (2015, July) Fleetnet - internet on the road. [Online]. Available: http://uk.nec.com/en_GB/emea/about/neclab_eu/projects/fleetnet.html

[11] ERTICO. (2015, July) Cvis. [Online]. Available: http://www.cvisproject.org/

[12] S. Al-Sultan, M. M. Al-Doori, A. H. Al-Bayatti, and H. Zedan, "A comprehensive survey on vehicular ad hoc network," *Journal of network and computer applications*, vol. 37, pp. 380–392, 2014.

[13] C2C-CC *et al.*, "Car 2 car communication consortium manifesto, overview of the c2c-cc system," 2007.

[14] A. Cardote, "Plataforma de comunicações veiculares com infraestrutura de suporte," Ph.D. dissertation, Instituto de Telecomunicações, Universidade Aveiro, 2014.

[15] Y. Zang, L. Stibor, B. Walke, H.-J. Reumerman, and A. Barroso, "Towards broadband vehicular ad-hoc networks-the vehicular mesh network (vmesh) mac protocol," in *Wireless Communications and Networking Conference, 2007. WCNC 2007. IEEE.* IEEE, 2007, pp. 417–422.

[16] D. Jiang and L. Delgrossi, "Ieee 802.11 p: Towards an international standard for wireless access in vehicular environments," in *Vehicular Technology Conference, 2008. VTC Spring 2008. IEEE.* IEEE, 2008, pp. 2036–2040.

[17] B. S. Gukhool and S. Cherkaoui, "Ieee 802.11 p modeling in ns-2," in *Local Computer Networks, 2008. LCN 2008. 33rd IEEE Conference on.* IEEE, 2008, pp. 622–626.

[18] "Trial-use standard for wireless access in vehicular environments (wave) - resource manager," *IEEE Std 1609.1-2006*, pp. 1–71, Oct 2006.

[19] "Ieee trial-use standard for wireless access in vehicular environments - security services for applications and management messages," *IEEE Std 1609.2-2006*, pp. 0_1–105, 2006.

[20] "Ieee standard for wireless access in vehicular environments (wave) - networking services," *IEEE Std 1609.3-2010 (Revision of IEEE Std 1609.3-2007)*, pp. 1–144, Dec 2010.

[21] "Ieee trial-use standard for wireless access in vehicular environments (wave) - multi-channel operation," *IEEE Std 1609.4-2006*, pp. 1–82, Nov 2006.

[22] M. Nekovee, "Sensor networks on the road: the promises and challenges of vehicular adhoc networks and vehicular grids," in *Proceedings of the Workshop on Ubiquitous Computing and e-Research*, 2005.

[23] H. Moustafa and Y. Zhang, *Vehicular networks: techniques, standards, and applications.* Auerbach publications, 2009.

[24] W. Kremer, "Realistic simulation of a broadcast protocol for an inter vehicle communication system (ivcs)," in *Vehicular Technology Conference, 1991. Gateway to the Future Technology in Motion., 41st IEEE.* IEEE, 1991, pp. 624–629.

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

[26] F. Warthman *et al.*, "Delay-and disruption-tolerant networks (dtns)," *A Tutorial. V.. 0, Interplanetary Internet Special Interest Group*, 2012.

[27] K. Fall and S. Farrell, "Dtn: an architectural retrospective," *Selected Areas in Communications, IEEE Journal on*, vol. 26, no. 5, pp. 828–836, 2008.

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

[29] F. Li and Y. Wang, "Routing in vehicular ad hoc networks: A survey," *Vehicular Technology Magazine, IEEE*, vol. 2, no. 2, pp. 12–22, 2007.

[30] P. R. Pereira, A. Casaca, J. J. Rodrigues, V. N. Soares, J. Triay, and C. Cervelló-Pastor, "From delay-tolerant networks to vehicular delay-tolerant networks," *Communications Surveys & Tutorials, IEEE*, vol. 14, no. 4, pp. 1166–1182, 2012.

[31] V. N. Soares, F. Farahmand, and J. J. Rodrigues, "Improving vehicular delay-tolerant network performance with relay nodes," in *Next Generation Internet Networks, 2009. NGI'09.* IEEE, 2009, pp. 1–5.

[32] R. C. Shah, S. Roy, S. Jain, and W. Brunette, "Data mules: Modeling and analysis of a three-tier architecture for sparse sensor networks," *Ad Hoc Networks*, vol. 1, no. 2, pp. 215–233, 2003.

[33] T. L. Willke, P. Tientrakool, and N. F. Maxemchuk, "A survey of inter-vehicle communication protocols and their applications," *Communications Surveys & Tutorials, IEEE*, vol. 11, no. 2, pp. 3–20, 2009.

[34] S. Guo, M. H. Falaki, E. A. Oliver, S. Ur Rahman, A. Seth, M. A. Zaharia, and S. Keshav, "Very low-cost internet access using kiosknet," *ACM SIGCOMM Computer Communication Review*, vol. 37, no. 5, pp. 95–100, 2007.

[35] H. Soroush, N. Banerjee, A. Balasubramanian, M. D. Corner, B. N. Levine, and B. Lynn, "Dome: a diverse outdoor mobile testbed," in *Proceedings of the 1st ACM International Workshop on Hot Topics of Planet-Scale Mobility Measurements.* ACM, 2009, p. 2.

[36] V. N. Soares, F. Farahmand, and J. J. Rodrigues, "A layered architecture for vehicular delay-tolerant networks," in *Computers and Communications, 2009. ISCC 2009. IEEE Symposium on.* IEEE, 2009, pp. 122–127.

[37] N. Benamar, K. D. Singh, M. Benamar, D. El Ouadghiri, and J.-M. Bonnin, "Routing protocols in vehicular delay tolerant networks: A comprehensive survey," *Computer Communications*, vol. 48, pp. 141–158, 2014.

[38] Y. Chen, C. Qiao, and X. Yu, "Optical burst switching: a new area in optical networking research," *Network, IEEE*, vol. 18, no. 3, pp. 16–23, 2004.

[39] A. Vahdat, D. Becker *et al.*, "Epidemic routing for partially connected ad hoc networks," Technical Report CS-200006, Duke University, Tech. Rep., 2000.

[40] J. Burgess, B. Gallagher, D. Jensen, and B. N. Levine, "Maxprop: Routing for vehicle-based disruption-tolerant networks." in *INFOCOM*, vol. 6, 2006, pp. 1–11.

[41] A. Balasubramanian, B. Levine, and A. Venkataramani, "Dtn routing as a resource allocation problem," *ACM SIGCOMM Computer Communication Review*, vol. 37, no. 4, pp. 373–384, 2007.

[42] T. Spyropoulos, K. Psounis, and C. S. Raghavendra, “Spray and wait: an efficient routing scheme for intermittently connected mobile networks,” in *Proceedings of the 2005 ACM SIGCOMM workshop on Delay-tolerant networking.* ACM, 2005, pp. 252–259.

[43] B. Hull, V. Bychkovsky, Y. Zhang, K. Chen, M. Goraczko, A. Miu, E. Shih, H. Balakrishnan, and S. Madden, “Cartel: a distributed mobile sensor computing system,” in *Proceedings of the 4th international conference on Embedded networked sensor systems.* ACM, 2006, pp. 125–138.

[44] S. Lahde, M. Doering, W.-B. Pöttner, G. Lammert, and L. Wolf, “A practical analysis of communication characteristics for mobile and distributed pollution measurements on the road,” *Wireless Communications and Mobile Computing*, vol. 7, no. 10, pp. 1209–1218, 2007.

[45] J. Ott and D. Kutscher, “A disconnection-tolerant transport for drive-thru internet environments,” in *INFOCOM 2005. 24th Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings IEEE*, vol. 3. IEEE, 2005, pp. 1849–1862.

[46] K. Scott, “Disruption tolerant networking proxies for on-the-move tactical networks,” in *Military Communications Conference, 2005. MILCOM 2005. IEEE.* IEEE, 2005, pp. 3226–3231.

[47] M. Gerla, C. Wu, G. Pau, and X. Zhu, “Content distribution in vanets,” *Vehicular Communications*, vol. 1, no. 1, pp. 3–12, 2014.

[48] M. Gerla, C. Lindemann, and A. Rowstron, “P2p manet’s-new research issues.” in *Peer-to-Peer Mobile Ad Hoc Networks*, 2005.

[49] B. Cohen, “Incentives build robustness in bittorrent,” in *Workshop on Economics of Peer-to-Peer systems*, vol. 6, 2003, pp. 68–72.

[50] A. Klemm, C. Lindemann, and O. P. Waldhorst, “A special-purpose peer-to-peer file sharing system for mobile ad hoc networks,” in *Vehicular Technology Conference, 2003. VTC 2003-Fall. 2003 IEEE 58th*, vol. 4. IEEE, 2003, pp. 2758–2763.

[51] S. Das, A. Nandan, and G. Pau, “Spawn: a swarming protocol for vehicular ad-hoc wireless networks,” in *Proceedings of the 1st ACM international workshop on Vehicular ad hoc networks.* ACM, 2004, pp. 93–94.

[52] A. Nandan, S. Das, G. Pau, M. Gerla, and M. Sanadidi, “Co-operative downloading in vehicular ad-hoc wireless networks,” in *Wireless On-demand Network Systems and Services, 2005. WONS 2005. Second Annual Conference on.* IEEE, 2005, pp. 32–41.

[53] K. Lee and I. Yap, “Cartorrent: A bit-torrent system for vehicular ad-hoc networks,” *Los Angeles*, 2006.

[54] C. Perkins, E. Belding-Royer, and S. Das, “Ad hoc on-demand distance vector (aodv) routing,” Tech. Rep., 2003.

[55] C. Rezende, A. Mammeri, A. Boukerche, and A. A. Loureiro, "A receiver-based video dissemination solution for vehicular networks with content transmissions decoupled from relay node selection," *Ad Hoc Networks*, vol. 17, pp. 1–17, 2014.

[56] H. Wu, R. Fujimoto, R. Guensler, and M. Hunter, "Mddv: a mobility-centric data dissemination algorithm for vehicular networks," in *Proceedings of the 1st ACM international workshop on Vehicular ad hoc networks.* ACM, 2004, pp. 47–56.

[57] Y. Zhang, J. Zhao, and G. Cao, "Roadcast: a popularity aware content sharing scheme in vanets," *ACM SIGMOBILE Mobile Computing and Communications Review*, vol. 13, no. 4, pp. 1–14, 2010.

[58] B. Yu and F. Bai, "Pyramid: Informed content reconciliation for vehicular peer-to-peer systems," 2015.

[59] R. H. Frenkiel, B. Badrinath, J. Borras, and R. D. Yates, "The infostations challenge: Balancing cost and ubiquity in delivering wireless data," *IEEE Personal Communications*, vol. 7, no. 2, pp. 66–71, 2000.

[60] W. H. Yuen, R. D. Yates, and S.-C. Mau, "Exploiting data diversity and multiuser diversity in noncooperative mobile infostation networks," in *INFOCOM 2003. Twenty-Second Annual Joint Conference of the IEEE Computer and Communications. IEEE Societies*, vol. 3. IEEE, 2003, pp. 2218–2228.

[61] A. Nandan, S. Tewari, S. Das, M. Gerla, and L. Kleinrock, "Adtorrent: Delivering location cognizant advertisements to car networks," in *WONS 2006: Third Annual Conference on Wireless On-demand Network Systems and Services*, 2006, pp. 203–212.

[62] C. Gkantsidis and P. R. Rodriguez, "Network coding for large scale content distribution," in *INFOCOM 2005. 24th Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings IEEE*, vol. 4. IEEE, 2005, pp. 2235–2245.

[63] R. Ahlswede, N. Cai, S.-Y. R. Li, and R. W. Yeung, "Network information flow," *Information Theory, IEEE Transactions on*, vol. 46, no. 4, pp. 1204–1216, 2000.

[64] M. Gerla and L. Kleinrock, "Vehicular networks and the future of the mobile internet," *Computer Networks*, vol. 55, no. 2, pp. 457–469, 2011.

[65] S. Ahmed and S. S. Kanhere, "Vanetcode: network coding to enhance cooperative downloading in vehicular ad-hoc networks," in *Proceedings of the 2006 international conference on Wireless communications and mobile computing.* ACM, 2006, pp. 527–532.

[66] U. Lee, J.-S. Park, J. Yeh, G. Pau, and M. Gerla, "Code torrent: content distribution using network coding in vanet," in *Proceedings of the 1st international workshop on Decentralized resource sharing in mobile computing and networking.* ACM, 2006, pp. 1–5.

[67] F. Bai and A. Helmy, "Impact of mobility on mobility-assisted information diffusion (maid) protocols," *Department Of Computer Science, USC," Technical Report*, 2005.

[68] J.-S. Park, M. Gerla, D. S. Lun, Y. Yi, and M. Médard, "Codecast: a network-coding-based ad hoc multicast protocol," *Wireless Communications, IEEE*, vol. 13, no. 5, pp. 76–81, 2006.

[69] T. Locher, P. Moor, S. Schmid, and R. Wattenhofer, "Free riding in bittorrent is cheap," in *Proc. Workshop on Hot Topics in Networks (HotNets)*. Citeseer, 2006, pp. 85–90.

[70] M. Grossglauser and D. Tse, "Mobility increases the capacity of ad-hoc wireless networks," in *INFOCOM 2001. Twentieth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, vol. 3. IEEE, 2001, pp. 1360–1369.

[71] Y.-C. Tseng, S.-Y. Ni, Y.-S. Chen, and J.-P. Sheu, "The broadcast storm problem in a mobile ad hoc network," *Wireless networks*, vol. 8, no. 2-3, pp. 153–167, 2002.

[72] A. Legout, G. Urvoy-Keller, and P. Michiardi, "Rarest first and choke algorithms are enough," in *Proceedings of the 6th ACM SIGCOMM conference on Internet measurement*. ACM, 2006, pp. 203–216.

[73] A. R. Bharambe, C. Herley, and V. N. Padmanabhan, "Analyzing and improving bittorrent performance," *Microsoft Research, Microsoft Corporation One Microsoft Way Redmond, WA*, vol. 98052, pp. 2005–03, 2005.

[74] P. Felber and E. W. Biersack, "Self-scaling networks for content distribution," in *Proc. International Workshop on Self-* Properties in Complex Information Systems*. Citeseer, 2004, pp. 1–14.

[75] T. Camp, J. Boleng, and V. Davies, "A survey of mobility models for ad hoc network research," *Wireless communications and mobile computing*, vol. 2, no. 5, pp. 483–502, 2002.

[76] C. Sommer and F. Dressler, "Progressing toward realistic mobility models in vanet simulations," *Communications Magazine, IEEE*, vol. 46, no. 11, pp. 132–137, 2008.

[77] F. J. Martinez, C. K. Toh, J.-C. Cano, C. T. Calafate, and P. Manzoni, "A survey and comparative study of simulators for vehicular ad hoc networks (vanets)," *Wireless Communications and Mobile Computing*, vol. 11, no. 7, pp. 813–828, 2011.

[78] J. Härri, F. Filali, and C. Bonnet, "Mobility models for vehicular ad hoc networks: a survey and taxonomy," *Communications Surveys & Tutorials, IEEE*, vol. 11, no. 4, pp. 19–41, 2009.

[79] D. Helbing, "Traffic and related self-driven many-particle systems," *Reviews of modern physics*, vol. 73, no. 4, p. 1067, 2001.

[80] T. Toledo, "Driving behaviour: models and challenges," *Transport Reviews*, vol. 27, no. 1, pp. 65–84, 2007.

[81] Institute of Transportation Systems. (2015, August) Sumo – simulation of urban mobility. [Online]. Available: http://www.dlr.de/ts/en/desktopdefault.aspx/tabid-9883/16931_read-41000/

[82] M. Behrisch, L. Bieker, J. Erdmann, and D. Krajzewicz, "Sumo–simulation of urban mobility," in *The Third International Conference on Advances in System Simulation (SIMUL 2011), Barcelona, Spain*, 2011.

[83] H. Conceição, L. Damas, M. Ferreira, and J. Barros, "Large-scale simulation of v2v environments," in *Proceedings of the 2008 ACM symposium on Applied computing.* ACM, 2008, pp. 28–33.

[84] PTV Group. (2015, August) Ptv vissim. [Online]. Available: http://vision-traffic.ptvgroup.com/en-us/products/ptv-vissim/

[85] G. D. Cameron and G. I. Duncan, "Paramics—parallel microscopic simulation of road traffic," *The Journal of Supercomputing*, vol. 10, no. 1, pp. 25–53, 1996.

[86] J. Härri, M. Fiore, F. Filali, and C. Bonnet. (2015, August) Vanetmobisim. [Online]. Available: http://vanet.eurecom.fr/

[87] University of Stuttgart. (2015, August) Canu mobility simulation environment (canumobisim). [Online]. Available: http://canu.informatik.uni-stuttgart.de/mobisim/index.html

[88] Scalable Network Technologies. (2015, August) Qualnet. [Online]. Available: http://web.scalable-networks.com/content/qualnet

[89] X. Zeng, R. Bagrodia, and M. Gerla, "Glomosim: a library for parallel simulation of large-scale wireless networks," in *Parallel and Distributed Simulation, 1998. PADS 98. Proceedings. Twelfth Workshop on.* IEEE, 1998, pp. 154–161.

[90] L. Bajaj, M. Takai, R. Ahuja, K. Tang, R. Bagrodia, and M. Gerla, "Glomosim: A scalable network simulation environment," *UCLA Computer Science Department Technical Report*, vol. 990027, p. 213, 1999.

[91] E. Giordano, R. Frank, G. Pau, and M. Gerla, "Corner: a realistic urban propagation model for vanet," in *Wireless On-demand Network Systems and Services (WONS), 2010 Seventh International Conference on.* IEEE, 2010, pp. 57–60.

[92] OpenSim Ltd. (2015, August) Omnet++. [Online]. Available: https://omnetpp.org/

[93] V. Andreas. (2015, August) Mixim. [Online]. Available: http://mixim.sourceforge.net/

[94] Information Sciences Institute of University of Southern California. (2015, August) The network simulator - ns-2. [Online]. Available: http://www.isi.edu/nsnam/ns/

[95] NSNAM. (2015, August) What is ns-3. [Online]. Available: https://www.nsnam.org/overview/what-is-ns-3/

[96] E. Weingärtner, H. Vom Lehn, and K. Wehrle, "A performance comparison of recent network simulators," in *Communications, 2009. ICC'09. IEEE International Conference on.* IEEE, 2009, pp. 1–5.

[97] Laboratory for Communications and Applications of École Polytechique Fédérale de Lausanne. (2015, August) Traffic and network simulation environment. [Online]. Available: http://lca.epfl.ch/projects/trans/

[98] M. Piorkowski, M. Raya, A. L. Lugo, P. Papadimitratos, M. Grossglauser, and J.-P. Hubaux, “Trans: realistic joint traffic and network simulator for vanets,” *ACM SIGMOBILE Mobile Computing and Communications Review*, vol. 12, no. 1, pp. 31–33, 2008.

[99] C. Sommer. (2015, August) Veins - the open source vehicular network simulation framework. [Online]. Available: http://veins.car2x.org/

[100] I. Framework. (2015, August) An open-source omnet++ model suite for wired, wireless and mobile networks. [Online]. Available: https://inet.omnetpp.org/

[101] MediaWiki. (2015, August) Traci. [Online]. Available: http://www.sumo.dlr.de/wiki/TraCI

[102] iTetris Project Consortium. (2015, August) itetris, the open simulation platform for intelligent transport system services. [Online]. Available: http://www.ict-itetris.eu/

[103] S.-Y. Wang and C.-C. Lin, “Nctuns 6.0: a simulator for advanced wireless vehicular network research,” in *Vehicular Technology Conference (VTC 2010-Spring), 2010 IEEE 71st.* IEEE, 2010, pp. 1–2.

[104] A. Keränen, J. Ott, and T. Kärkkäinen, “The one simulator for dtn protocol evaluation,” in *Proceedings of the 2nd international conference on simulation tools and techniques.* ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2009, p. 55.

[105] P. Hui, A. Chaintreau, J. Scott, R. Gass, J. Crowcroft, and C. Diot, “Pocket switched networks and human mobility in conference environments,” in *Proceedings of the 2005 ACM SIGCOMM workshop on Delay-tolerant networking.* ACM, 2005, pp. 244–251.

[106] N. Eagle and A. Pentland, “Reality mining: sensing complex social systems,” *Personal and ubiquitous computing*, vol. 10, no. 4, pp. 255–268, 2006.

[107] Dartmouth.

[108] S. Jain, K. Fall, and R. Patra, *Routing in a delay tolerant network.* ACM, 2004, vol. 34, no. 4.

[109] Delay-Tolerant Networking Research Group. (2015, August) Code. [Online]. Available: https://sites.google.com/site/dtnresgroup/home/code

[110] A. Keränen and J. Ott, “Opportunistic network environment simulator - special assignment report, helsinki university of technology,” *Department of Communications and Networking*, 2008.

[111] Netlab. (2015, August) The one, the opportunistic network environment simulator. [Online]. Available: http://www.netlab.tkk.fi/tutkimus/dtn/theone/

[112] A. Lindgren, A. Doria, and O. Schelen, “Probabilistic routing in intermittently connected networks,” in *Service Assurance with Partial and Intermittent Resources.* Springer, 2004, pp. 239–254.

[113] A. Lindgren, A. Doria, E. Davies, and S. Grasic, “Probabilistic Routing Protocol for Intermittently Connected Networks,” RFC 6693 (Experimental), Internet Engineering Task Force, Aug. 2012. [Online]. Available: http://www.ietf.org/rfc/rfc6693.txt

[114] M. Demmer, E. Brewer, K. Fall, M. Ho, R. Patra, M. Demmer, E. Brewer, K. Fall, S. Jain, M. Ho, and R. Patra, “Implementing delay tolerant networking,” Tech. Rep., 2003.

[115] Delay-Tolerant Networking Research Group (DTNRG). (2015, July) Prophet router. [Online]. Available: http://info.n4c.eu/code/DTN2/file/0aaf7724519d/doc/manual/ro_prophet.html

[116] M. Demmer and K. Fall, “Dtlsr: delay tolerant routing for developing regions,” in *Proceedings of the 2007 workshop on Networked systems for developing regions*. ACM, 2007, p. 5.

[117] Delay-Tolerant Networking Research Group (DTNRG). (2015, July) Dtlsr router. [Online]. Available: http://info.n4c.eu/code/DTN2/file/0aaf7724519d/doc/manual/ro_dtlsr.html

[118] ——. (2015, July) Flood router. [Online]. Available: http://info.n4c.eu/code/DTN2/file/0aaf7724519d/doc/manual/ro_flood.html

[119] ——. (2015, July) tca-router routing algorithm. [Online]. Available: http://info.n4c.eu/code/DTN2/file/0aaf7724519d/doc/manual/ro_tca-router.html

[120] ——. (2015, July) External router. [Online]. Available: http://info.n4c.eu/code/DTN2/file/0aaf7724519d/doc/manual/ro_external.html

[121] ——. (2015, July) Static router. [Online]. Available: http://info.n4c.eu/code/DTN2/file/0aaf7724519d/doc/manual/ro_static.html

[122] S. Burleigh, “Interplanetary overlay network: An implementation of the dtn bundle protocol,” in *Consumer Communications and Networking Conference, 2007. CCNC 2007. 4th IEEE*, 2007, pp. 222–226.

[123] ——, “Compressed Bundle Header Encoding (CBHE),” RFC 6260 (Experimental), Internet Engineering Task Force, May 2011. [Online]. Available: http://www.ietf.org/rfc/rfc6260.txt

[124] M. Ramadas, S. Burleigh, and S. Farrell, “Licklider Transmission Protocol - Specification,” RFC 5326 (Experimental), Internet Engineering Task Force, Sep. 2008. [Online]. Available: http://www.ietf.org/rfc/rfc5326.txt

[125] S.-A. Lenas, S. C. Burleigh, and V. Tsaoussidis, “Bundle streaming service: design, implementation and performance evaluation,” *Transactions on Emerging Telecommunications Technologies*, 2013.

[126] S. Schildt, J. Morgenroth, W.-B. Pöttner, and L. Wolf, “Ibrdtn: A lightweight, modular and highly portable bundle protocol implementation,” in *Electronic Communications of the EASST*. Citeseer, 2011.

[127] OpenWRT. (2015, July) Openwrt - wireless freedom. [Online]. Available: https://www.openwrt.org/

[128] D. Ellard, R. Altmann, A. Gladd, and D. Brown, "Dtn ip neighbor discovery (ipnd)," Working Draft, IETF Secretariat, Internet-Draft draft-irtf-dtnrg-ipnd-02, November 2012, https://tools.ietf.org/id/draft-irtf-dtnrg-ipnd-02.txt. [Online]. Available: https://tools.ietf.org/id/draft-irtf-dtnrg-ipnd-02.txt

[129] M. Demmer, J. Ott, and S. Perreault, "Delay-Tolerant Networking TCP Convergence-Layer Protocol," RFC 7242 (Experimental), Internet Engineering Task Force, Jun. 2014. [Online]. Available: http://www.ietf.org/rfc/rfc7242.txt

[130] H. Kruse and S. Ostermann, "Udp convergence layers for the dtn bundle and ltp protocols," Working Draft, IETF Secretariat, Internet-Draft draft-irtf-dtnrg-udp-clayer-00, November 2008, http://www.ietf.org/internet-drafts/draft-irtf-dtnrg-udp-clayer-00.txt. [Online]. Available: http://www.ietf.org/internet-drafts/draft-irtf-dtnrg-udp-clayer-00.txt

[131] Haxx. (2015, August) libcurl - the multiprotocol file transfer library. [Online]. Available: http://curl.haxx.se/libcurl/

[132] "Ieee standard for information technology - telecommunications and information exchange between systems - local and metropolitan area networks - specific requirement part 15.4: Wireless medium access control (mac) and physical layer (phy) specifications for low-rate wireless personal area networks (wpans)," *IEEE Std 802.15.4a-2007 (Amendment to IEEE Std 802.15.4-2006)*, pp. 1–203, 2007.

[133] SQLite Consortium. (2015, August) Sqlite. [Online]. Available: https://www.sqlite.org/

[134] A. Broder and M. Mitzenmacher, "Network applications of bloom filters: A survey," *Internet mathematics*, vol. 1, no. 4, pp. 485–509, 2004.

[135] Viagénie. (2015, July) Postellation project. [Online]. Available: http://postellation.viagenie.ca/

[136] M. Blanchet, S. Perreault, and J.-P. Dionne, "Postellation: an enhanced delay-tolerant network (dtn) implementation with video streaming and automated network attachment." SpaceOps, 2012.

[137] B. Moon, J. Miner, and Jan. (2015, July) Jdtn. [Online]. Available: http://sourceforge.net/projects/jdtn/

[138] M. B. S. Prasad, M. W. Arshad, S. A. Shahzad, S. Z. Sajjad, H. Eriksson, B. Aminian, A. Doria, B. Pehrson, and M. Zennaro, "Bytewalla 3."

[139] Department of Communications and Networking, Helsinki University of Technology. (2015, July) Dt-talkie. [Online]. Available: http://www.netlab.tkk.fi/tutkimus/dtn/dttalkie/

[140] Delay-Tolerant Networking Research Group (DTNRG). (2015, July) Dtn2. [Online]. Available: https://sites.google.com/site/dtnresgroup/home/code/dtn2documentation

[141] Networking for Communications Challenged Communities, "D2.2: Functional specification for dtn infrastructure software comprising rfc 5050 bundle agent and associated components," Tech. Rep. 1.2.

[142] S. Symington, S. Farrell, H. Weiss, and P. Lovell, "Bundle Security Protocol Specification," RFC 6257 (Experimental), Internet Engineering Task Force, May 2011. [Online]. Available: http://www.ietf.org/rfc/rfc6257.txt

[143] A. Seth, P. Darragh, S. Liang, Y. Lin, and S. Keshav, "An architecture for tetherless communication," *Disruption Tolerant Networking*, vol. 5142, 2005.

[144] Jet Propulsion Laboratory, California Institute of Technology. (2015, July) Ion, interplanetary overlay network. [Online]. Available: https://ion.ocp.ohiou.edu/

[145] S. Burleigh, "Interplanetary overlay network (ion) design and operation," *Jet Propulsion Laboratory, California Institute of Technology, v1*, vol. 8.

[146] ——, "Contact graph routing," 2010.

[147] Institut für Betriebssysteme und Rechnerverbund der TU Braunschweig. (2015, July) Ibr-dtn - a modular and lightweight implementation of the bundle protocol. [Online]. Available: https://trac.ibr.cs.tu-bs.de/project-cm-2012-ibrdtn

[148] M. Doering, S. Lahde, J. Morgenroth, and L. Wolf, "Ibr-dtn: an efficient implementation for embedded systems," in *Proceedings of the third ACM workshop on Challenged networks*. ACM, 2008, pp. 117–120.

[149] Network Architectures and Protocols. (2015, August) Network architectures and protocols. [Online]. Available: http://nap.av.it.pt/

[150] B. Tavares, "Transmissão oportunística de informação em redes veiculares," Master's thesis, Departamento de Engenharia Eletrónica Telecomunicações e Informática, Universidade de Aveiro, 2013.

[151] L. Guedes, "Transmissão oportunística de informação em redes veiculares," Master's thesis, Departamento de Engenharia Eletrónica Telecomunicações e Informática, Universidade de Aveiro, 2014.

[152] JSON. (2015, August) Introducing json. [Online]. Available: http://json.org/

[153] M. Perillo, "Network coding overview."

[154] T. Ho, M. Médard, R. Koetter, D. R. Karger, M. Effros, J. Shi, and B. Leong, "A random linear network coding approach to multicast," *Information Theory, IEEE Transactions on*, vol. 52, no. 10, pp. 4413–4430, 2006.

[155] MathWorks. (2015, August) Matlab. [Online]. Available: http://www.mathworks.com/products/matlab/

[156] C. Carlson and D. Clay, "The earth model–calculating field size and distances between points using gps coordinates," *Site Specific Management Guidelines*, p. 4, 1999.

[157] Python. (2015, September) Python. [Online]. Available: https://www.python.org/

[158] iMatix Corporation. (2015, October) Zmq. [Online]. Available: http://zeromq.org/

[159] Wikipedia. (2015, September) Memory-mapped file. [Online]. Available: https://en.wikipedia.org/wiki/Memory-mapped_file

[160] OpenWRT. (2015, October) Openwrt build system – usage. [Online]. Available: http://wiki.openwrt.org/doc/howto/build

[161] SOURCEFORGE.NET. (2015, November) Ptpd. [Online]. Available: http://ptpd.sourceforge.net/

[162] Network Time Foundation. (2015, November) Ntp: The network time protocol. [Online]. Available: http://www.ntp.org/

[163] Aldum. (2015, October) Opkg package manager. [Online]. Available: http://wiki.openwrt.org/doc/techref/opkg

[164] Porto Digital. (2015, October) Porto digital - o projecto. [Online]. Available: http://www.portodigital.pt/

[165] How-To Geek. (2015, September) Understanding the load average on linux and other unix-like systems. [Online]. Available: http://www.howtogeek.com/194642/understanding-the-load-average-on-linux-and-other-unix-like-systems/

[166] Scout. (2015, September) Understanding linux cpu load - when should you be worried? [Online]. Available: http://blog.scoutapp.com/articles/2009/07/31/understanding-load-averages

Appendix A

How-Tos

How-To compile from HelixEmu to Netrider boards

Gonçalo Pessoa

October 13, 2015

Abstract

Instructions to compile the source code from the HelixEmu project to the Netrider boards.

There are also described some specific characteristics of the source code that were implement in order to deploy the content distribution schemes in the boards.

This document is a continuation of the emulator101 written by Ricardo Dias where a brief description of the HelixEmulator is given.

1 Compiling

The HelixEmulator source code can be compiled in two ways:

1. To run has an emulator in a machine (with linux-based operating system);
2. To run directly the Helix in a laptop (with linux-based operating system).

Both approaches are described in emulator101. However, to compile to the Netrider boards a set of modifications and considerations has to be done.

These modifications can be divided as follows:

- Modification of the CMakeLists.txt file (if new files were added to the HelixEmu project).
- Modification of the Helix original Makefile.
- Modification of the Helix original Makefile of OpenWRT.

The CMakeList.txt file located in `dev/Helix/src/` has to be modified in order to add new modules. For example, if a new module called Logging has to be added to the HelixEmu source code this information must be introduced in the CMakeList.txt file. This should be done inside of the scope `SET( helixcore_SRC...)`. Without this adding the cmake will not be able to generate a correct Makefile to compile these new modules.

The Helix makefile located in `dev/Helix/src/` must be modified in order to handle the new modules added to implement the content distribution strategies.

Thus, the macros `LIBS`, `CPPFLAGS`, `SRCS_C`, and `SRCS_CPP` must be modified in order to add additional libraries, compilation flags, and new file to the compilation process. To compile the HelixEmu source code with the new content distribution schemes the following `CPPFLAGS` were added:

- `-std=c++0x` since libraries of a newer C++ standard are used.
- `-D_GLIBCXX_USE_NANOSLEEP` to use sleep functions with a precision of nanoseconds.

Likewise in the CMakeList.txt file the new files must be added to the compile process whereby the `SRCS_CPP` (and, if new C files were added, the `SRCS_C`) must be modified.

The working Helix Makefile is in Appendix A and the CMakeLists.txt in Appendix B.

Once modified the CMakeLists.txt and Makefile the HelixEmu must be compiled. This can be done following the next steps:

1. Change the user location to the `Helix/build/` directory;
2. Clean the folder's content;
3. Generate the Makefile using the cmake with the non-emulator option selected.

   ```
   cmake -DEMULATOR=off ../src
   ```

4. Compile

   ```
   make clean all
   ```

Now the Helix is ready to run in a laptop (the binary file is located in the `Helix/bin/` folder), but not in the Netrider boards. The next procedure in the cross-compiling of the code to the Netrider architecture and operating system.

To cross-compile the code is used the build-root of the VeniamOS (operating system based in the OpenWRT which was modified by Veniam and is the one used in the Netrider boards). The process of cross-compilation in done using a Makefile located in `VeniamOS/openwrt/package/Helix/` (if this folder does not exists it must to be created). A working version of this Makefile is provided in Appendix C. This Makefile is responsible to cross-compile a package provided by the user that must be located in the `VeniamOS/openwrt/dl/` folder as a `tar.bz2` file. The name and the extension of it must be in compliance with the information of the OpenWRT Helix Makefile. This `tar.bz2` must be a compression of the `dev/Helix/` folder which is the folder where the source code (`Helix/src/`), the binary files (`Helix/bin/`), etc, are located.

Before compiling the Helix the support library must be also compiled and integrated in the VeniamOS build-root. The procedure is the same, a `tar.bz2` of the `dev/libhelix/` must be created and transfer to the `VeniamOS/openwrt/dl/` folder. The process of cross-compilation in done using a Makefile located in

`VeniamOS/openwrt/package/libhelix/` (if this folder does not exists it must to be created). A working version of this Makefile is provided in Appendix D.

Once created and transfered all the files and Makefiles the compilation can be performed following the next steps:

1. Change to the source directory of VeniamOS (`VeniamOS/openwrt/`);

2. Select the linhelix and Helix packages in the configuration menu of VeniamOS

   ```
   make menuconfig
   ```

   The libhelix package is located in `Libraries/`

   The Helix package is located in `Network/`

3. Compile the libhelix (if it is already compiled and none modification was performed there is no need to compile it again)

   ```
   make package/libhelix/{clean,compile} V=99
   ```

4. Compile the Helix

   ```
   make package/Helix/{clean,compile} V=99
   ```

These procedures result in two ipk files that can be installed in the Netrider boards. These are generated during the compiling process and saved in the following folder: `VeniamOS/openwrt/bin/ar71xx/packages/`.

Once transfered the ipk files to the board the libhelix and Helix programs can be installed as follows:

- `opkg install libhelix***.ipk`
- `opkg install Helix***.ipk`

If there is errors related, for example, with a downgrading of the libhelix or Helix versions the already installed packages must be removed.

- `opkg list`
- `opkg remove libhelix***`
- `opkg remove Helix***`

Run content distribution experiments in the netrider boards

Gonçalo Pessoa

October 13, 2015

Abstract

Concerns and warnings regarding the running of a content distribution experiment in the netrider boards.

1 Compiling

The compilation procedure is explained in a document called "compileFromHelixEmuToNetrider".

To compile to the boards the Makefile must be generated with the `DEMULATOR` flag `off`. Thus, only the source code that are outside of the `EMULATOR` define scope will be compiled.

2 Concerns

2.1 Set log folder, log file name, and generate packets

As is described in the thesis document, all the packets that will be disseminated must be generated before the initiating of the Helix program (which is done in the `src/helix.cpp` file).

Thus, the first thing to be done is the setting of the Log folder where the Logging module writes the log data of the experiment. In the `Logging` class there are two variables to define the log folder path and the log file name. When the program is running in the boards the log path is always the same as defined in the configuration file of Helix (`helix.conf`). On the other hand, the log file name can be defined at will.

Never forget that all the paths (api, storage, and log) defined in the `helix.conf` must exist in the board, otherwise an error will occur at the starting of Helix.

After that, the packets must be generated and the structures associated with the content distribution strategies must be also initiated and filled as follows.

```
// Generate packets in RSUs (before start process)
unsigned NumberOfPackets = 320; // Very dependent of
    MAX_PAGES macro in storageDisk.c
```

```
uint8_t fileID = 1;

// If node is an RSU (NODE_TYPE=2)
if(get_node_type() == 2) {
dbg(INFO,"[MAIN]␣Generating␣packets␣in␣RSUs␣...\n");

for (int i=0; i<NumberOfPackets; i++) {

        // Inite HelixHeader of the packet
        HelixHeader helix_header=HelixHeader();

           helix_header_init(&helix_header);
        helix_header.expiryDate = time(NULL) + 3600*4;
        helix_header.srcEID = get_endpoint_id();
        helix_header.dstEID = DTN_EID_ALL_NODES;
        helix_header.serviceID =
           DTN_CONTENT_DIST_3_SERVICE_ID;
        helix_header.numNeigh = 1;
        helix_header.fileID = fileID;
        helix_header.totalPacketsOfFile =
           NumberOfPackets;
        helix_header.optionsLength = 0;

        // Generate payload of ~32KB
        uint32_t maxSize = DTN_MAX_BUF_SIZE - sizeof(
           HelixHeader);
        char * buf = new char[maxSize];
        memset(buf,'z',maxSize);
        helix_header.dataLength = maxSize;

        // Create packet
        Packet* p = Packet::make(helix_header, buf);

        // Brute-force hash (this is not neeeded)
        p->header.hash = i+1;

        // Push it to storage (expiry table)
        storage->push(p);


        // Initiate content dist structures
        switch (get_routing_version()) {

        case 2: {
        // Init linkListLNHF
        NodeLNHF node_tmp;
```

```
node_tmp.hash = p->header.hash;
node_tmp.nHops = helix_header.numNeigh;
node_tmp.nTx = 0;
node_tmp.isOnHold = false;

// Add node to LNHF structures
pthread_mutex_lock(&Routing::
   link_list_LNHF_mutex);
routing->linkListLNHF.push_front(node_tmp);

   pthread_mutex_unlock(&Routing::
   link_list_LNHF_mutex);

// Sort LNHF structures
pthread_mutex_lock(&Routing::
   link_list_LNHF_mutex);
routing->linkListLNHF.sort(listCompRSU);
pthread_mutex_unlock(&Routing::
   link_list_LNHF_mutex);

break;
}
case 3: {
// Evaluate if it is the last element (if yes,
    sort main list)
// This is done to decrease the number of sort
    of main list structure
bool sortMainList = false;
if (i+1 == NumberOfPackets)
sortMainList = true;

// Insert empty node into structures
dbg(INFO,"[MAIN]␣Inserting␣empty␣node␣into␣
   structures␣(%u,%u,%u)...\n",p->header.
   fileID, p->header.totalPacketsOfFile, p->
   header.hash);
int retInsert = Routing::handlerLRBF.
   insertEmptyNode(p->header.fileID, p->header
   .totalPacketsOfFile, p->header.hash,
   sortMainList);

// Update MapFiles table (because the node
   pushed a new packet)
dbg(INFO,"[MAIN]␣Updating␣MapFiles␣table␣(
   because␣the␣node␣pushed␣a␣new␣packet)...\n"
   );
```

```
        int retInc = Routing::handlerLRBF.
            incnPacketsStored(p->header.fileID);

        break;
        }

        case 4: {
        // Not implemented to the boards but the
            source code can be found in the
            correponding emulator scope
        break;

        case 5: {
        // Do nothing
        break;
        }
        default: {
        break;
        }
        }

        delete p;
        delete buf;

        // Logging
        Routing::log.inc_packets_stored_total();
        Routing::log.inc_packets_stored_per_timestamp
            ();
        Routing::log.
            inc_packets_recv_good_per_timestamp();
}
```

2.2 Setting macros related with the content dissemination process

There are a set of macros that must be taken into account when the content distribution process is running in the boards:

- MIN_CONT_ADV_PERIOD and MAX_CONT_ADV_PERIOD - these macros define the advertisement periodicity in seconds and are located in src/Routing/Routing.h;
- TIME_STEP_LOG - this macro defines the logging periodicity in nanoseconds and is located in src/Routing/Logging.h;
- ELEMENT_VALID_TIME - this macro defines the time in seconds that a certain element is valid in the listPairs (or listRankNeighs) and is located in src/Routing/HandlerLRBF.h (and in src/Routing/HandlerLRGF.h);

- MIN_REFRESH_PERIOD and MAX_REFRESH_PERIOD - these macros define the refreshment periodicity in seconds and are located in src/Routing/HandlerLRBF.h and src/Routing/HandlerLRGF.h;

The values specified by the laboratory experiment were the following:

- MIN_CONT_ADV_PERIOD = 4
- MAX_CONT_ADV_PERIOD = 5
- TIME_STEP_LOG = 10000000
- ELEMENT_VALID_TIME = 30
- MIN_REFRESH_PERIOD = 29
- MAX_REFRESH_PERIOD = 31

2.3 Logging of performance metrics

The support bash script used to lagging information related with the performance metrics (load, cpu usage and memory) is provided in appendix. This script must be executed as follows:

```
bash ***/performance/metrics/board.sh "PERFORMANCE LOGFILE NAME"
```

2.4 Running

To run the helix in boards for a content distribution experiment the following command should be executed:

```
/usr/sbin/helix -c -f ***/helix.conf
```

The helix.conf used in the laboratory tests was the following:

```
{
"EID": "***"
"socket_port": "4556",
"storage_path": "/root/GP/DTN/Storage/",
"storage_cap": "20",
"api_path": "/root/GP/DTN/api/",
"log_path": "/root/GP/DTN/Log",
"log_output": "2",
"log_level": "2",
"rt_version": "3",
"11p_iface": [ "wlan1" ]
}
```

Another important parameter is the board type which is defined in the /root/type file. This information will be used along the source code of Helix to identify the type of the node.

3 Statistical analysis

It is also provided a MATLAB script which performs a statistical analysis of the collected log data. Be careful using it since it works based on a specific format of log folder and file names, nodes number.