Universidade de Aveiro
2019

Departamento de
Eletrónica, Telecomunicações e Informática

João Bernardo
Castanheira Patrício

Network Mechanisms for Swarms of Drones in Aquatic Sensing Environments

Mecanismos de Rede para Swarms de Drones em Ambientes de Monitorização Aquática

Universidade de Aveiro Departamento de
Eletrónica, Telecomunicações e Informática
2019

João Bernardo
Castanheira Patrício

Network Mechanisms for Swarms of Drones in Aquatic Sensing Environments

Mecanismos de Rede para Swarms de Drones em Ambientes de Monitorização Aquática

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Engenharia Eletrónica e Telecomunicações, realizada sob a orientação científica do Doutor Nuno Miguel Abreu Luís, Investigador Auxiliar do Instituto de Telecomunicações, e co-orientação da Doutora Susana Isabel Barreto de Miranda Sargento, Professora Catedrática do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro.

o júri / the jury

presidente / president	**Doutor Paulo Miguel Nepomuceno Pereira Monteiro** Professor Associado da Universidade de Aveiro
vogais / examiners committee	**Doutor Pedro Miguel Salgueiro dos Santos** Professor Auxiliar Convidado da Faculdade de Engenharia da Universidade do Porto
	Doutor Nuno Miguel Abreu Luís Investigador Auxiliar do Instituto de Telecomunicações (Orientador)

agradecimentos / acknowledgements

A concretização desta etapa deve-se não só a mim, mas a todos aqueles que me rodeiam e fazem parte da minha vida.
Em primeiro lugar, quero agradecer aos meus pais, não só pelo apoio implacável ao longo de todo o meu percurso académico, mas também pela educação que me forneceram, pela responsabilidade que sempre me incutiram e pelos enormes esforços que sempre fizeram por mim sem cessar. Depois, agradecer também a toda a minha restante família, nomeadamente ao meu irmão por me aturar até durante a vida académica, à minha tia Adelina pelo apoio constante, e à minha avó Idalina por todo o carinho e boas energias que sempre me transmitiu.
Quero agradecer a todos os meus amigos, aos de longa data, aos que a vida académica me proporcionou e aos que encontrei no grupo de investigação NAP nesta última etapa. Sem eles todo este percurso teria sido uma monotonia sem vida.
Um agradecimento especial ao Doutor Miguel Luís por toda a orientação e apoio perspicaz ao longo deste trabalho, que foram essenciais para a sua conclusão. Agradecer, igualmente, à Professora Doutora Susana Sargento pelo acolhimento neste grupo de investigação e apoio no desenvolvimento desta dissertação.
Por fim, agradecer à Universidade de Aveiro pelo ensino de qualidade proporcionado e ao Instituto de Telecomunicações pela formação extra em investigação que possibilitou ao longo deste percurso. A todos, que direta ou indiretamente contribuíram para que chegasse a este momento, um sincero obrigado.

Resumo

Com o desenvolvimento de plataformas inteligentes que permitem monitorizar vários ambientes, os drones apresentam-se como um recurso fundamental capaz de responder às mais vastas aplicações. A monitorização de meios aquáticos com recurso a drones é uma destas aplicações e a comunicação entre os mesmos torna-se um aspeto fundamental, tanto em tarefas de navegação como em tarefas de sensorização.
Testar um ambiente aquático com um elevado número de drones aquáticos é muito caro, requer muito tempo e vários recursos, por isso, plataformas de simulação tornam-se elementos de grande importância. Nesta dissertação é desenvolvido um simulador, com uma arquitetura modular, tendo por base uma rede tolerante a atrasos, sendo capaz de simular ambientes aquáticos o mais semelhante possível a ambientes aquáticos reais.
Para além do simulador desenvolvido, esta dissertação propõe métodos e estratégias de formação de clusters de drones, permitindo que os drones aquáticos elejam, de uma forma distribuída, os gateways de cada cluster que serão responsáveis por encaminhar os dados recolhidos pelos drones em direção à estação em terra. Foram implementados dois métodos de eleição de gateway, um focado na energia dos drones aquáticos, e outro capaz de considerar diferentes métricas, tais como a qualidade de ligação, a centralidade e a energia. Os métodos propostos foram avaliados através de vários cenários em que os clusters são construídos e alterados de forma dinâmica, e foi observado que a escolha de gateways com um método baseado em várias métricas, e juntamente com uma estratégia de controlo apropriada, proporciona um melhor comportamento da rede ao longo das tarefas de monitorização aquática.

Abstract

With the development of intelligent platforms for environment sensing, drones present themselves as a fundamental resource capable of responding to the widest range of applications. Monitoring aquatic sensing environments is one such application and the communication between them becomes a key aspect for both navigation and sensing tasks.
Testing an aquatic environment with a high number of Unmanned Surface Vehicles (USVs) is very costly, requiring a lot of time and resources. Therefore, simulation platforms become elements of great importance . In this dissertation a simulator is developed containing a modular architecture, based on a delay tolerant network, being capable of simulating aquatic environments as similar as possible to real aquatic environments.
In addition to the developed simulator, this dissertation presents methods and strategies of cluster formation, allowing the aquatic drones to select, in a distributed way, the gateways of each cluster that will be responsible for forwarding collected data towards the gateway on land. Two gateway selection methods were implemented, one focused on the energy of aquatic drones, and one considering different metrics such as link quality, centrality and energy. The proposed methods were evaluated across several cases and scenarios, with clusters built and changed in a dynamic way, and it was observed that the election of gateways with a method based on several metrics, together with appropriated control strategy, provides a better outcome of the network behaviour throughout the aquatic monitoring tasks.

Contents

List of Figures

List of Tables

Acronyms

API	Application Programming Interface
AquaticDTNS	Aquatic Surface Delay Tolerant Networks Simulator
ARR	Acquitted Reception Rate
ASL	Asymmetry level
WMEWMA	Window Mean with Exponential Weighted Moving Average
BS	Base Station
CH	Cluster Head
CLA	Convergence Layer Adapter
CPU	Central Processing Unit
DCU	Decentralized Unit
DTN	Delay-Tolerant Network
DTN2	Delay-Tolerant Network 2
E2E	End-to-End
ETX	Expected Transmission Count
F-LQE	Fuzzy Link-Quality Estimator
GPU	Graphics Processing Unit
GW	Gateway
HEED	Hybrid Energy-Efficient Distributed Clustering
IBR-DTN	IBR Delay Tolerant Network
ID	Identification
IEEE	Institute of Electrical and Electronics Engineers
IoT	Internet of Things
IP	Internet Protocol
KB	Kilobyte

LEACH	Low Energy Adaptive Clustering Hierarchy
LoRa	Long Range LPWAN Technology
LPWAN	Low Power Wide-Area Network
LQE	Low Quality Estimators
LQI	Link Quality Indicator
MAC	Medium Access Control
MANET	Mobile Ad-Hoc Networks
MAE	Mean Absolute Error
mOVE	Mobile Opportunistic Vehicular
NAP	Network Architecture and Protocols
NIC	Network Interface Controller
ns-3	Network Simulator 3
ONE	Opportunistic Network Environment
OPS	Opportunistic Protocol Simulator
OSI	Open System Interconnection
PAmuLQE	Passive Multi-hop Link Quality Estimator
PRoPHET	Probabilistic Routing Protocol using History of Encounters and Transitivity
PRR	Packet Reception Ratio
PSR	Packet Success Ratio
RE-TOPSIS	Reliability Technique for Order of Preference by Similarity to Ideal Solution
ROS	Robot Operating System
RSSI	Received Strength Indicator
RX	Reception
SCF	Store-Carry-and-Forward
SF	Stability Factor
SN	Sensor Node
SSI	Signal Strength Indicator
SNR	Signal-to-Noise Ratio
TCP	Transmission Control Protocol
UDP	User Datagram Protocol

USV	Unmanned Surface Vehicle
UUV	Unnamed Underwater Vehicle
WAVE	Wireless Access for Vehicular Environments
WiFi	WiFi IEEE 802.11a/b/g
WNN-LQE	Wavelet-Neural-Network-based Link Quality Estimator
WSN	Wireless Sensor Networks

Chapter 1

Introduction

Internet of Things (IoT) is turning our Cities into smart Cities by offering opportunities to use sensed data to manage traffic, reduce pollution, make better use of infrastructure and keep citizens safe. With a growing awareness on the climate change and their following impacts in the present and near future, monitoring our environment is a big contribute to answer such situation, and it is also part of the technological revolution we live in.

Being the planet Earth 70% made of water, monitoring large areas of water, as aqua-cultures, rivers, etc., requires an effort that is humanly hard to fulfill, due to constrains such as access to the site, human effort, life threatening situations, cost, low efficiency and many others. New ways of monitoring and exploring these sites are of great interest. Therefore, Unmanned Surface Vehicle (USV), like aquatic drones, have been gaining ground for patrol and monitoring of aquatic environments [8]. These USVs are able to help in cases such as monitoring aquatic environments and patrol coastal areas, gathering information for marine biology studies, ocean warming, among many others. For such tasks, several USVs are required with constant communication within each other, so that relevant information reaches the infrastructure on earth. Therefore, these clusters of USVs must be able to self-organize and decide who among them will gather and send their relevant information to the Base Station (BS).

USVs within the clusters are mobile, which means that these clusters of aquatic drones will change their position over time, and the connectivity and link quality between the USVs will change frequently. In order to maintain a successful delivery ratio of data to the BS, the USVs must have a dynamic and decentralized architecture, with a low latency, bandwidth efficiency, and high adaptability, for an unstable network [3]. On clusters with an increasing number of USVs in range, a multi-hop network will emerge without a need for a centralized authority, where any node can enter and leave this network, loosing communication with the destination, and even enter another network of another swarm of USVs. In such scenarios, the concept of Delay Tolerant Networks DTN is usually adopted.

The USVs' mobility may decrease the time for communication and data gathering within the network. In order to keep the best quality possible of the communication, the USVs must find and use the links and communication routes with the best quality, improving the performance of the network. However, such characterization demands for a process of a link quality estimation (LQE) capable of increasing the network performance without increasing the network overhead. Due to the network mobility, the loss of connection to the Gateway (GW) may happen when a USV goes on a mission alone or travels between clusters. The GW is responsible for transmitting information from a cluster to the BS, sometimes using another network interface technology than the one used to communicate within the cluster's network. Thus, a GW must be elected within each cluster and, as the network of each cluster changes, the election process must occur over time. Such process could consider a wide range of features and characteristics, such as energy, centrality and many others.

This dissertation aims to provide strategies capable of making each USV self-adapt to the cluster they are in and its network, delivering the collected information to the BS through the elected GW, keeping the lowest cost and overhead possible. Testing such environment with several clusters and USVs would be very costly and require a lot of time and resources. For that reason, a simulator capable of testing existing and proposed network strategies was developed, while keeping an environment similar to the reality.

1.1 Objectives

The main goal of this dissertation is the study of mechanisms of communication on multi-cluster self-adaptive aquatic USVs and the implementation of gateway election strategies, for a better performance in aquatic monitoring. Therefore, the objectives of this dissertation can be enumerated as follows:

- Design, implement and evaluate a simulator capable of replicating the aquatic monitoring scenarios, with a modular architecture that enables the implementation of different network mechanisms;
- Study, implement and evaluate forwarding LQE-based strategies over DTNs for mobile environments;
- Design, implement and evaluate different dynamic Gateway election strategies in a multi-cluster environment;
- Evaluate the functionality and overall performance of the developed network solution all together.

1.2 Contributions

In this scope, the main contributions of this dissertation are:

- The development of a modular network simulator prepared for DTN scenarios with forwarding strategies through the best quality path implemented;
- A passive neighborhood evaluation for GW election;
- A rank equation for GW election based on several metrics;
- A passive GW election within each cluster, where each USV elects the GW without injecting extra packets in the network;
- A self-adaptive architecture that enables the USVs to auto-recognize a new GW, either when they are already within a cluster or when they enter a new one.

1.3 Document Structure

The remainder of this document is organized as follows:

- **Chapter 2** - *State of the Art* - This chapter presents a general analysis and description of the fundamental concepts related to this dissertation: aquatic USVs, delay tolerant networks, LQE-based strategies and GW election;

- **Chapter 3** - *A DTN Simulator for Swarms of Drones* - This chapter describes the simulator architecture, the message flowing within the simulator architecture , the configuration of the simulations and the simulator's graphic interface;
- **Chapter 4** - *Network Mechanisms* - This chapter provides the implementation of the forwarding strategies and gateway election schemes proposed;
- **Chapter 5** - *Tests and Results* - This chapter presents the simulator scenarios and evaluates the implemented solutions;
- **Chapter 6** - *Conclusions and Future Work* - This chapter presents the dissertation's conclusions along with suggestions for possible improvements and future work.

Chapter 2

State of the Art

This chapter aims to provide the reader with fundamental concepts required to understand the work presented in this dissertation. Section 2.1 introduces the concept of smart cities and presents the concept of DTN and an architectural overview. Section 2.3 presents the concept and mechanisms of Link Quality Estimators, lists a set of studies related to the routing and forwarding strategies in DTNs, and lists a set of studies related to the gateway/cluster head election mechanisms. Section 2.4 presents a set of network simulation platforms and frameworks. Section 2.5 presents the chapter summary and considerations.

2.1 Smart Environments

Nowadays, we live in an era where our cities are turning into smart cities. Such is a consequence of the technological revolution we live in, where everything from vehicles to buildings are embedded with software, sensors and actuators, all equipped with communication devices that connect them to a smart platform. Such connection enables a new myriad of applications in Internet-of-Things (IoT) [9]. From these applications, a smart city can increase operational efficiency, share information with the public and improve both the quality of services and well being of the citizens.

Almeida et al. [1] described a large city-wide architecture where the aquatic monitoring platform takes part, as illustrated in Figure 2.1. The main components of this architecture are heterogeneous nodes, such as Data Collecting Units (DCUs), cars, bicycles, aerial drones and USVs.

The aquatic platform that is integrated in this architecture has specific requirements that demands an extended architecture [3]. Those extensions are:

- Heterogeneous mobile nodes (USVs) with short range communication (e.g. WiFi) and/or with long range communication (e.g. LoRa);
- Collection and dissemination of environmental data, from the short range communication mobile nodes to the long range ones (mobile gateways);
- Monitoring an entire aquatic tank with the minimal cost;
- Fallback support of mobile aerial gateways if USVs are isolated.

In order for the heterogeneous mobile nodes (USVs) to send their data, from the monitoring of the aquatic environment, to be gathered by the cluster's Head/Gateway and successfully reach the Base Station (BS), while taking into account the network characteristics on such environment

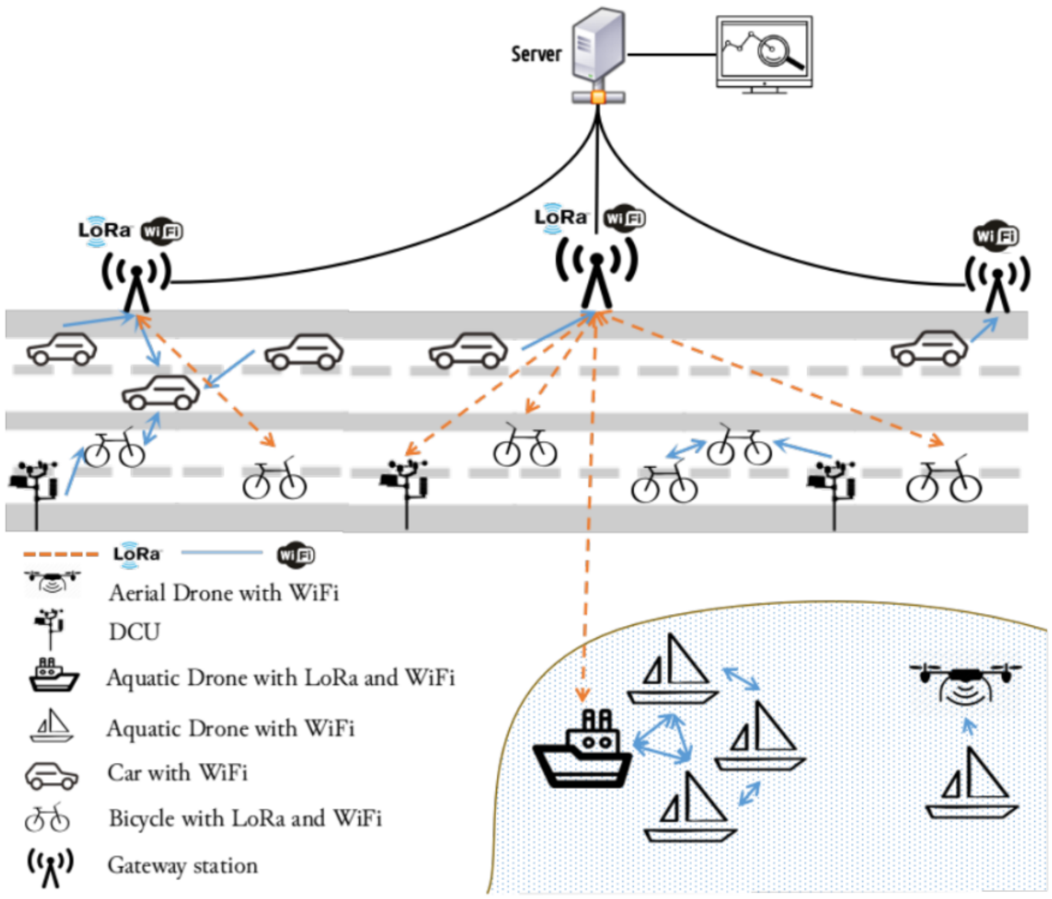


Figure 2.1: City-Wide Platform Overview [1].

---

with mobility and unstable topology, they all should be built upon a Delay Tolerant Network (DTN) architecture where the nodes can store packets, to handle the low connectivity and the high delay of the communication.

2.2 Delay Tolerant Networks

A Delay Tolerant Network (DTN) is a network with an architecture able to deal with long propagation delays, low data rates and intermittent connectivity [1]. It was firstly created to assure communication in extreme environments, like spatial and interplanetary communications [10]. Due to its characteristics, it has been used in many terrestrial applications where wireless networks face challenging conditions for communication, such as wireless military networks, sparse wireless sensor networks and vehicular networks. Figure 2.2 summarizes possible applications for a DTN.

2.2.1 Challenges and Requirements

The majority of the protocols used in the Internet are based on the Transmission Control Protocol (TCP)/Internet Protocol (IP), which does not work well in environments with intermittent connectivity since it does not comply with the fundamental assumptions built into the Internet: existence of an end-to-end path between source and destination, small probability of end-to-end packet drop, communication reliability, etc. [2]. Mobile Ad-Hoc Networks (MANET)

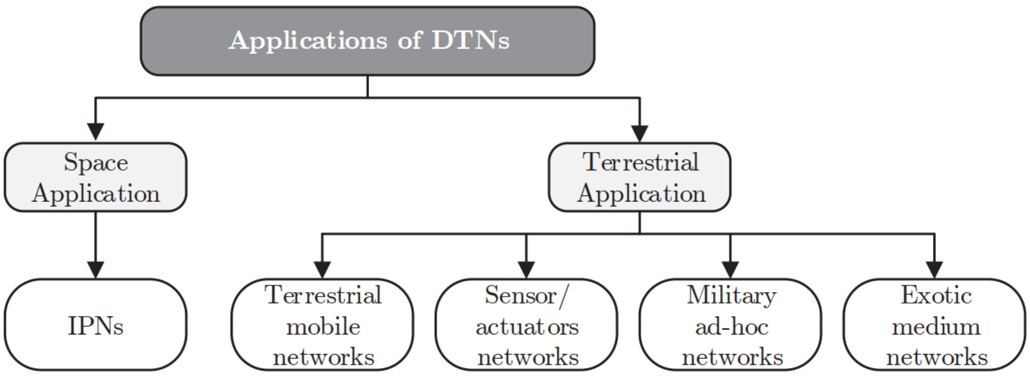


Figure 2.2: DTN applications taxonomy [2].

face challenging environments that do not hold upon the referred protocol assumptions. These challenging characteristics are [11]:

- Intermittent Connectivity: in challenging environments connectivity issues lead to an unreliable End-to-End (E2E) path. A DTN supports communication without a clearly defined E2E path.
- Variable Delay: long propagation delays and variable queuing delays, as nodes contribute to delays on the E2E path, which can defeat protocols that depend on a quick return of acknowledgements or data.
- Asymmetric Data Rates: conventional protocols used on the Internet support moderate asymmetries of bidirectional data rate, but for considerable asymmetric values, these protocols lean to not working properly.
- High Error Rates: bit errors on links require correction, increasing the network traffic due to the transmission of packets and processing. For the same link-error rate, the number of needed retransmissions is smaller when the communication is hop-by-hop than end-to-end.

DTN protocols aim to overcome the appointed challenging characteristics.

2.2.2 Architecture

The DTN architecture was designed to adapt itself to network connection disruptions, while providing a support framework for network functionalities, such as transport and routing. DTN uses naming, layering, encapsulation, and persistence storage to interconnect heterogeneous portions of a network.

The DTN architecture introduces a new layer that bridges the Application layer to the Transport Layer: the Bundle layer [12]. In Figure 2.3, the differences between the Open System Interconnection (OSI) stack, the TCP/IP stack and the DTN stack model are represented.The Bundle Layer allows the network to cope with intermittent connectivity by implementing a hop-by-hop reliability mechanism and optional E2E acknowledgement, consequently allowing the network to have persistent storage. As illustrated in Figure 2.4, the Bundle Forwarder is the central element of the DTN's architecture, being responsible for forwarding the packets between the storage, the Convergence Layer Adapter (CLA) and the applications, depending on the routing decisions.

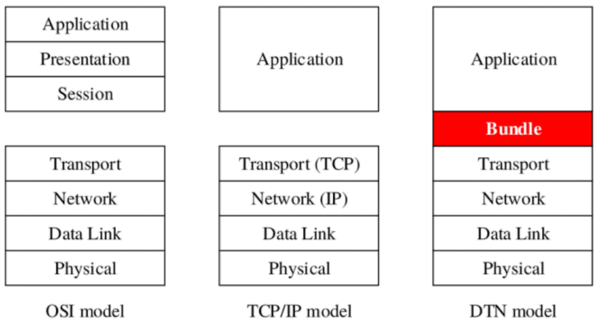


Figure 2.3: Comparison between OSI, TCP/IP and DTN models [3].

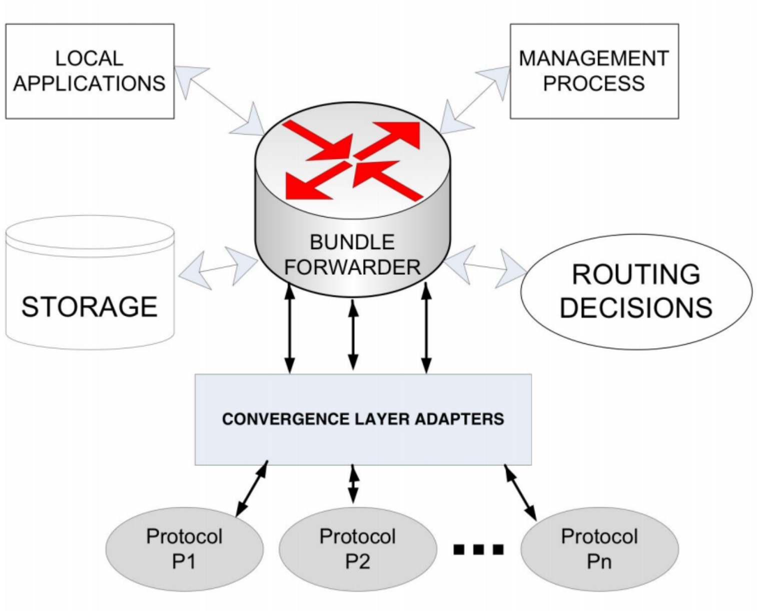


Figure 2.4: DTN's conceptual architecture [4].

A DTN can use different protocols for data delivery, such as TCP/IP and Ethernet, because it has a module denominated Convergence Layer Adapter (CLA) that is responsible for providing all required methods to transport the DTN data (also referred as bundles) to the respective protocols [4].

Store-Carry-and-Forward Mechanism

To overcome previously presented challenges, DTNs use a mechanism called Store-Carry-and-Forward (SCF), illustrated in Figure 2.5, where a DTN node receives a data packet, stores it, and then forwards it as soon as the communication is available [13]. The communication between nodes in the DTN architecture can be unavailable for a large period of time, or a link disruption can occur during the transmission of information, in which case the sender node must preserve the information for posterior retransmission. For that, DTN nodes must be equipped with a storage device. In short, the SCF mechanism plays an important role to minimize the problems caused by unreliable connectivity.

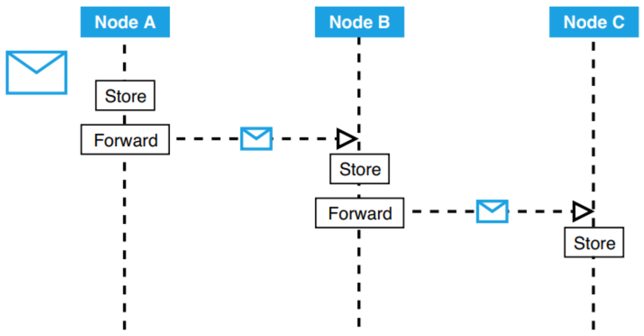


Figure 2.5: DTN's Store-Carry-and-Forward Mechanism [3].

There are several implementations of DTNs developed, being DTN2 [14] and IBR-DTN [15] some of the most known. From previous works, several problems were identified when using these implementations, such as not being robust in environments with high mobility, and introducing high unnecessary complexity for the intended applications [2] [3].

2.2.3 Mobile Opportunistic Vehicular Architecture (mOVE)

mOVE [16] is a DTN software developed and implemented by the Network Architecture and Protocols (NAP)[1] research group. This software was developed in a C/C++ language and was designed to be highly modular and extensible. It supports, for communication, WiFi technology IEEE 802.11a/b/g, IEEE 802.11p, Wireless Access in Vehicular Environments (WAVE) and LoRa technology.

In Figure 2.6 it is presented an overall architecture of mOVE. It is composed by nine modules: Neighboring, Socket, Reception (RX), Application Programming Interface (API) Management, Storage, Routing (Decision Maker), Sensor Listener, DCU Listener, and LoRa Socket, all with logging capabilities. A description over these architectural modules is presented next:

Neighboring: This module is responsible for the discovery of neighboring nodes. Each node advertises periodically its presence by broadcasting a *neighbor announcement* packet. When such packet is received, the node updates its internal neighboring table with the new set of information.

Socket and RX: These modules are responsible for processing the incoming and outgoing data or control packets. The Socket module is an UDP socket that provides an abstraction layer to send/receive packets to/from neighboring nodes. The RX module is constantly checking if any data was received and, when it does, it classifies the packets by their type and forwards them to the neighboring module (*neighbor announcements*) and routing module (*data packets*).

API Management: This module allows the mOVE node to interact with external applications through the use of mOVE packets. It also contains a registration service that provides access to the mOVE Apps.

Storage: This module is responsible for storing several packets and relevant information for the forwarding decision. Due to its robustness, it is able to deal with unexpected power outages. Its most important methods are *push/pull* a packet to/from the *storage*; verify the destination EID and get a copy of the packet (*peek*), *serviceID* or expiration time; and to remove or consult

[1]www.it.pt/Groups/Index/36

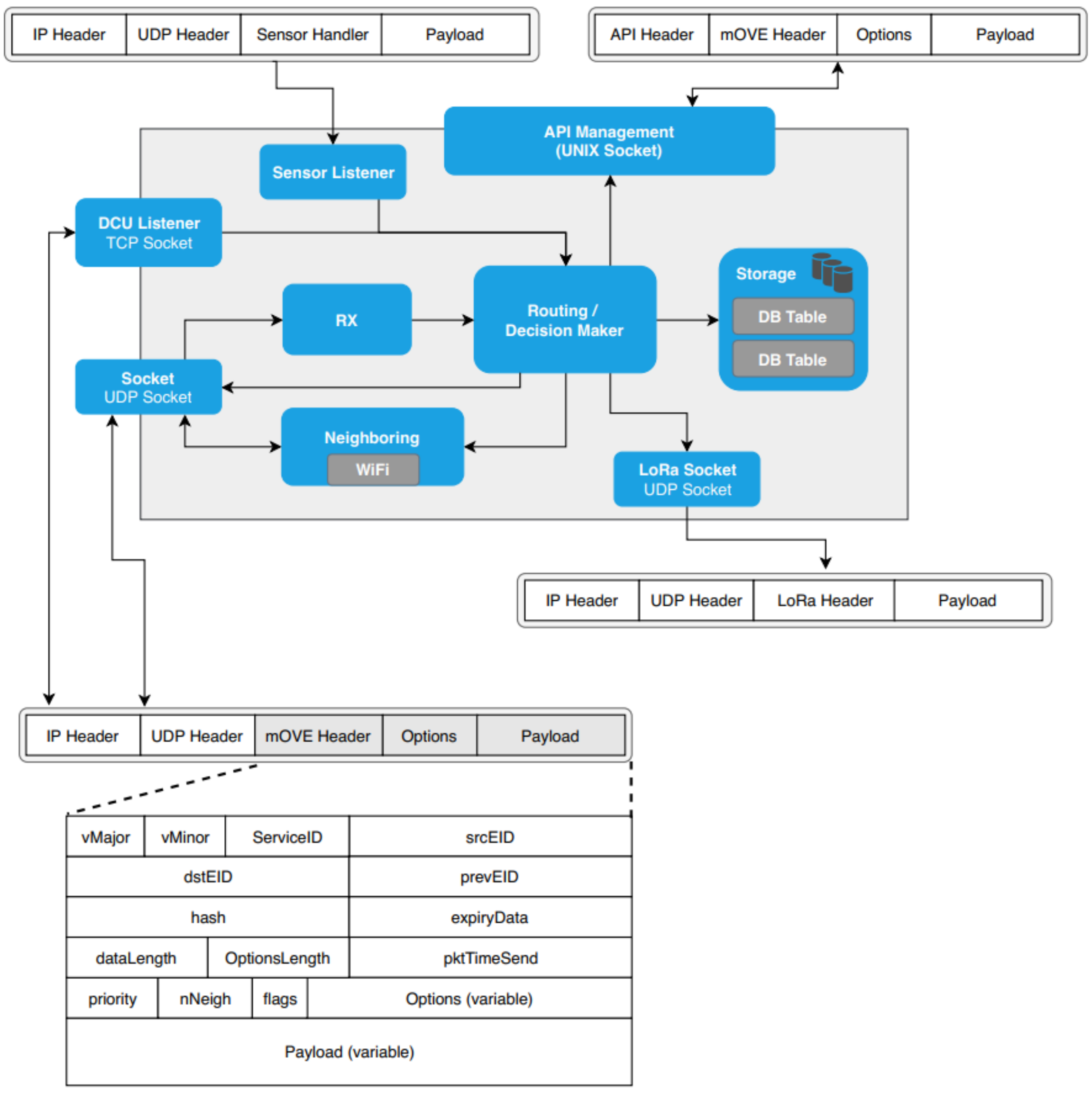


Figure 2.6: mOVE architecture [3].

the existence of a specific packet in the *storage*. This module contains four tables to store packets:

- *Expiry*: Packets sorted by expiration time;
- *OnHold*: Packets awaiting to be retransmitted, sorted by time on hold;
- *Own*: Packets which are meant for this node, sorted by service EID;
- *NoData*: Table with *hash* information about the known packets, sorted by expiration time.

LoRa Socket: This module is responsible for forwarding the chosen data to be delivered through LoRa technology.

Sensor Listener: All the data collected by the node's sensors is forward by this socket to the mOVE platform.

DCU Listener: This module establishes a connection to a DCU, creating a channel between the mobile node and the DCU.

Routing: This module makes decisions of which packets should be sent, in which order, to which neighbors, based on the routing strategy. To that end, it aims to minimize the replicas in the network and the packets maintained in *Storage*.

mOVE does not follow the reference specification of the *Bundle Layer* described in RFC 5050 [4], instead, the *bundles* are called *mOVE Packets*. As presented in Figure 2.6, the *packet* structure is the following:

- **mOVE Header**
  - vMajor and vMinor: mOVE Version;
  - Service ID;
  - srcEID and dstEID: Source and Destinations EIDs
  - prevEID: Previous custodian EID;
  - Hash (unique identifier for a packet);
  - Expiry Data (time of creation + lifetime);
  - Payload Length;
  - Options Length;
  - Priority;
  - nNeigh: Current number of neighbors that received the packet;
  - flags: To identify the packets type (e.g E2E ACK, Delivered);
  - pktTimeSend: Packet's last temporal reference in which the packet was forwarded;
- **Options**: optional field that can be added later (e.g. list of neighbours where the packet was received).
- **Payload**: array of bytes with a maximum of 32KB (if options were added the maximum size would decrease).

mOVE presents a complete but simple and straight forward DTN architecture. For so, the DTN architecture implemented upon the simulator developed by this dissertation was based on the mOVE's modules Api Management, Routing, Storage and Neighboring.

2.3 Data Gathering Mechanisms

In order for the relevant data to be delivered at the BS with efficiency, the node should have a good knowledge about the connection quality to the surrounding nodes, so it can forward the relevant data properly, and have a knowledge of which is the network Gateway that will deliver the gathered data to the BS.

2.3.1 Link Quality Estimators

In networks using hardware that transmit data through radio signals, it is important to monitor the quality of the connection between devices for optimization on routing decisions. Link Quality Estimators (LQE), like Received Signal Strength Indicator (RSSI), Signal-to-Noise Ratio (SNR), and Link Quality Indicator (LQI) are considered hardware based estimators because they do not require any additional computation, being directly read from the transceiver. But, accordingly to several studies, hardware based estimators are inaccurate [17]. This inaccuracy is caused by the measurement of particular symbols on only successfully received packets. On the other hand, software based estimators are able to account the reception ratio or the average number of packet transmissions/retransmissions [3].

Several previous works presented software based estimators. Baccour et al. [17] considered Packet Reception Ratio (PRR) that counts, on the receiver side, the average number of packet retransmissions required before a successful reception, just like Acquitted Reception Rate (ARR) does, but on the sender side. These are quite simple and recommended for applications that require low complexity level with moderate performance. Window Mean with Exponentially Weighted Moving Average (WMEWMA) [18] and Kalman filter based LQE [19] approximate the packet reception rate. The first applies filtering on the PRR metric to smooth it, so that this new metric resists to transient fluctuation of PRRs, while continuing responsive to major quality changes. The second one approximates the packet reception ratio based on RSSI and pre-calibrated PRR/SNR curve [20]. Expected Transmission Count (ETX) [21] estimates the number of transmissions needed to send a packet successfully, and Four-Bit Wireless Link Estimation [22] approximates the average number of packet transmissions/retransmissions before a successful reception. However, they heavily depend on the tuning of their parameters.

More effective LQEs have been developed, based on more than a single link property. Baccour et al. [23] proposed the Fuzzy Link-Quality Estimator (F-LQE) based on four quality properties: Packet Success Ratio (PSR), link Asymmetry level (ASL), Stability Factor (SF) and channel quality (SNR). This fuzzy rule returns the membership of the link in the fuzzy subset of good links. Despite being faster, fuzzy methods incur a loss of precision. Wavelet-Neural-Network-based Link Quality Estimation (WNN-LQE) presented by Sun et al. [24] uses PRR and SNR, but due to its complex process for the estimation, its implementation in Wireless Sensor Networks (WSN) of nodes is not trivial.

In a swarm of aquatic drones, where the network is used for environmental monitoring and data dissemination, turning the network into a dense one, an active monitoring on the quality of the links may not be feasible because of the increase of load and latency. Daniela et al. [25] proposed the Passive Multi-hop Link Quality Estimator (PAmuLQE) based on: i) the estimated Signal Strength Indicator (SSI) between two nodes, and ii) the bit rate computed with the observed delay on each received packet. This strategy does not increase the network overhead and energy consumption for the USV, as each USV spreads its adjacency graph with the LQE classification, using the neighboring announcements that each USV already sends, making this a passive monitoring. The weight factor that is added in the adjacency graph is computed as follows [3]:

$$Weight = (1 - Link\_Quality) \times 100, \tag{2.1}$$

where *Link_ Quality* is given by

$$Link\_Quality = (1 - \beta) \times SSI\_est\_norm + \beta \times \frac{\sum_{i=0}^{Npackets-1} BitRate\_norm(i) \times Age\_factor(i)}{\sum_{i=0}^{Npackets-1} Age\_factor(i)}. \tag{2.2}$$

The *SSI_ est_ norm* is the estimated normalized value of SSI (given by the distance between the two nodes). The *BitRate_ norm* represents the normalized bit rate, and is given by

$$BitRate(i) = \frac{pkt\_size(i)}{pkt\_delay(i)}. \tag{2.3}$$

The factor β is expressed between 0 and 1, and represents a confidence level on each estimation

$$\beta = \frac{\sum_{i=0}^{Npackets-1} Age\_factor(i)}{Npackets}, \tag{2.4}$$

where *Age_factor* is given by

$$Age\_factor(i) = 1 - \frac{min[(time\_now - time\_received(i)), MAX\_AGE]}{MAX\_AGE}, \tag{2.5}$$

where *time_ received(i)* is the time that the packet i was received by the node. The older the packet is, the lower the age factor is. The *MAX_ AGE* is the maximum age allowed for a packet to be kept in the DTN storage.

In our developed work a link quality based estimator is implemented based on the presented PAmuLQE, but with a few modifications on the link quality of Equation 2.2.

2.3.2 Forwarding Strategies

Routing is the process of choosing the path that the information should go to a specific destiny in the network. Traditional routing protocols for wired and wireless networks fail in challenging environments, so different routing protocols must be used on DTNs. Several protocols have been developed for different situations, using different factors for the routing decision, such as shortest path, safest path, and many others. Some of these routing protocols are very simple, but introduce a high overhead on the network, others are more complex, requiring higher resources to process. Some relevant routing protocols are presented next.

Direct delivery [26] is a single copy routing protocol with no need of knowledge about the network to make forwarding decisions. In this protocol the source node carries a message until it meets the final destination. This results on a very minimum use of bandwidth and resources, but a very high delivery delay, sometimes even infinite.

Epidemic [5] is a multi-copy protocol that floods the bundle through the network. After exchanging summary vectors to avoid replications (Figure 2.7), and consulting a list of recently established contacts, the source floods the bundle to every neighbor that does not have it. It also has a hop count to prevent infinite message replication. This protocol increases the network overhead as the packets can reach the destination by multiple paths, but minimizes the delivery delay, maximizing the delivery ratio.

PRoPHET (Probabilistic Routing Protocol using History of Encounters and Transitivity) is a protocol that estimates the delivery probability based on the history of encounters, considering that some network nodes create connectivity patterns that are not completely random over time, consequently having some predictability. For that, it applies a probabilistic metric, *delivery predictability*, calculated through the history of node encounters and transitivity. If the *delivery predictability* of the bundle destination is higher at the potential node receptor, a bundle is

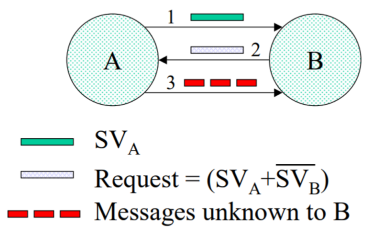


Figure 2.7: Message exchange in the Epidemic Routing Protocol [5].

replicated and sent. This protocol introduces additional overhead, in order to keep information about the estimation of meeting probabilities [27]. Some recent works have provided different variations of the PRoPHET protocol. Almeida in [28] proposed Q-PRoPHET, a quality-based routing protocol that adds multi-hop decisions based on the path's qualities. The LQE used is based on the RSSI and link stability, which turns it into an active monitoring estimator.

PAmuLQE (LQE Passive multi-hop Link Quality Estimator) is a routing protocol developed by Daniela et al. [3] where each node is able to evaluate all network links in the team. All nodes spread their adjacency graph with the LQE classifications through the neighbor announcements, making this process a passive monitoring. It has three variants (Figure 2.8) regarding the acknowledgment process:

- Passive multi-hop LQE with broadcast end-to-end acknowledgement (PAmuLQE-B-E2E): in this variant the destination node sends the acknowledgment packet in broadcast to all its neighbors. If the same packet is transmitted to a neighboring node that has the knowledge that this packet has already been delivered, this neighbor will send and acknowledge the packet in unicast to the sender node;

- Passive multi-hop LQE with unicast end-to-end acknowledgement (PAmuLQE-U-E2E): in this variant the end-to-end acknowledgement is sent like a data packet in unicast. When the packet is retransmitted from the sender to the gateway, it is deleted from a relay node every time that the next hop transmission is acknowledged;

- Passive multi-hop LQE with neighbor acknowledgement (PAmuLQE-NACK): in the third variant each node keeps the packet only until it receives the acknowledgement from the next node. This means that there is only one copy of the packet in the network at each time.

Table 2.1 summarizes the differences between the presented forwarding strategies. As it is presented, the Direct contact strategy is the simplest but less efficient one. The Epidemic strategy introduces a very high replication rate, while the Q-PRoPHET and the PAmuLQE strategies have a lower replication rate, which results on a lower overhead in the network. These later two strategies can have a single or multiple copy of a packet throughout the network, and are both passive. But PAmuLQE not only achieves a lower replication rate of its packets, as it has a truly passive monitoring without overhearing, or having other characteristics that can lead it to become an active monitoring. Such features lead the PAmuLQE to introduce less overhead in the network, when compared with the previous forwarding strategies. This will be used in the work developed in this thesis.

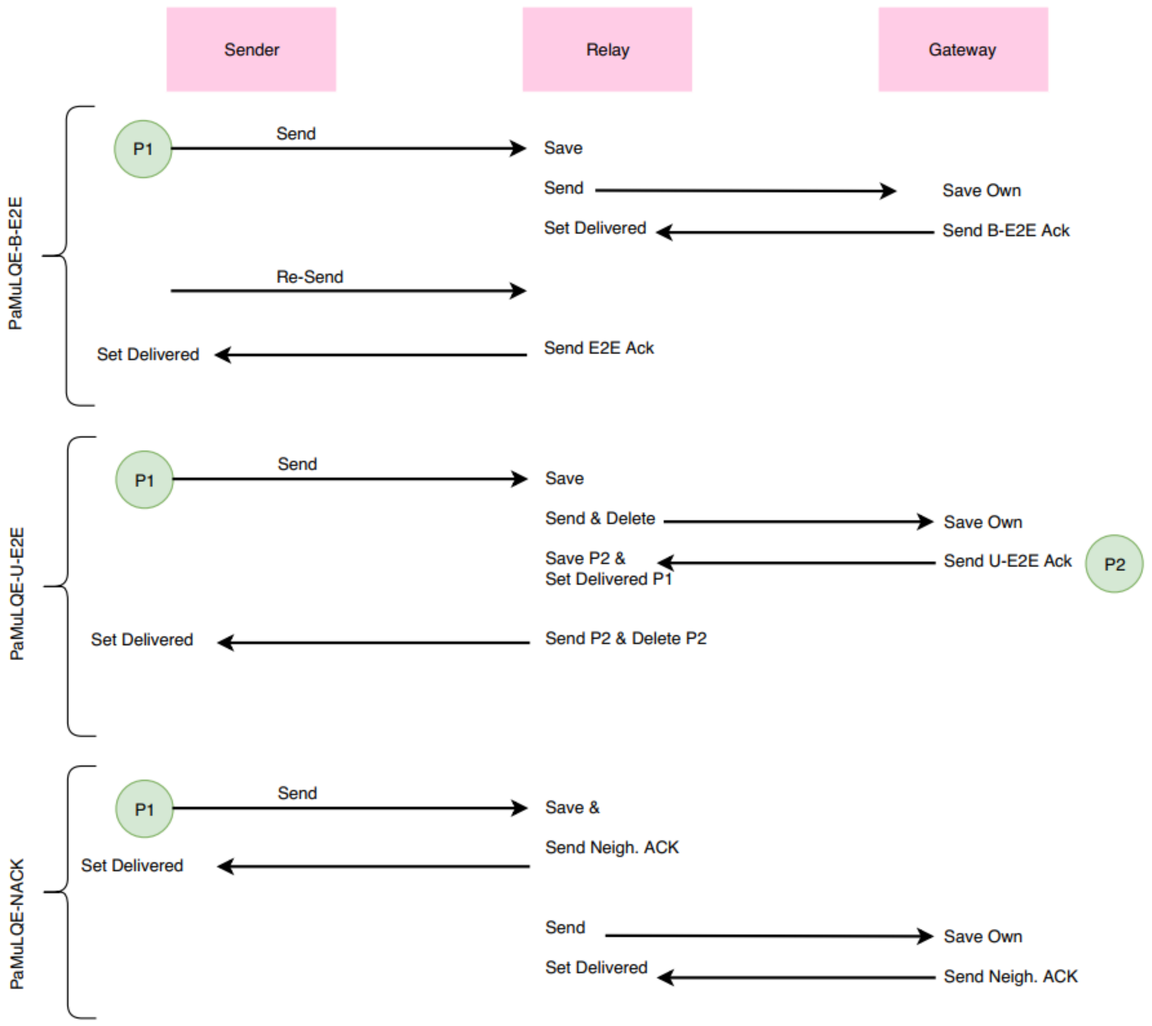


Figure 2.8: Routing logic for the three variants of the link quality-based routing strategy [3].

Name	**Type**	**Single/Multiple copy**	**Replication Rate**	**Passive/Active Monitoring**
Direct contact	Direct	S	N/A	N/A
Epidemic	Flooding	M	Very High	N/A
Q-PRoPHET	Probabilistic	S/M	Medium	Passive with Overhearing
PAmuLQE	Probabilistic	S/M	Low/Medium	Passive

Table 2.1: Forwarding Strategies overview.

2.3.3 Gateway Election Schemes

In order to increase the lifetime and quality of a network, a common approach is the clustering of the network. It consists on rearranging or dividing the network into a hierarchy of groups according to a specified weight function, and then set a node to be the Gateway, also referred as Cluster Head (CH) (Figure 2.9) [29]. In our work, a cluster is a team of nodes that share the same gateway and have multi-hop path between each other.

The network nodes have limited power supply, as they are battery-operated, so it is important

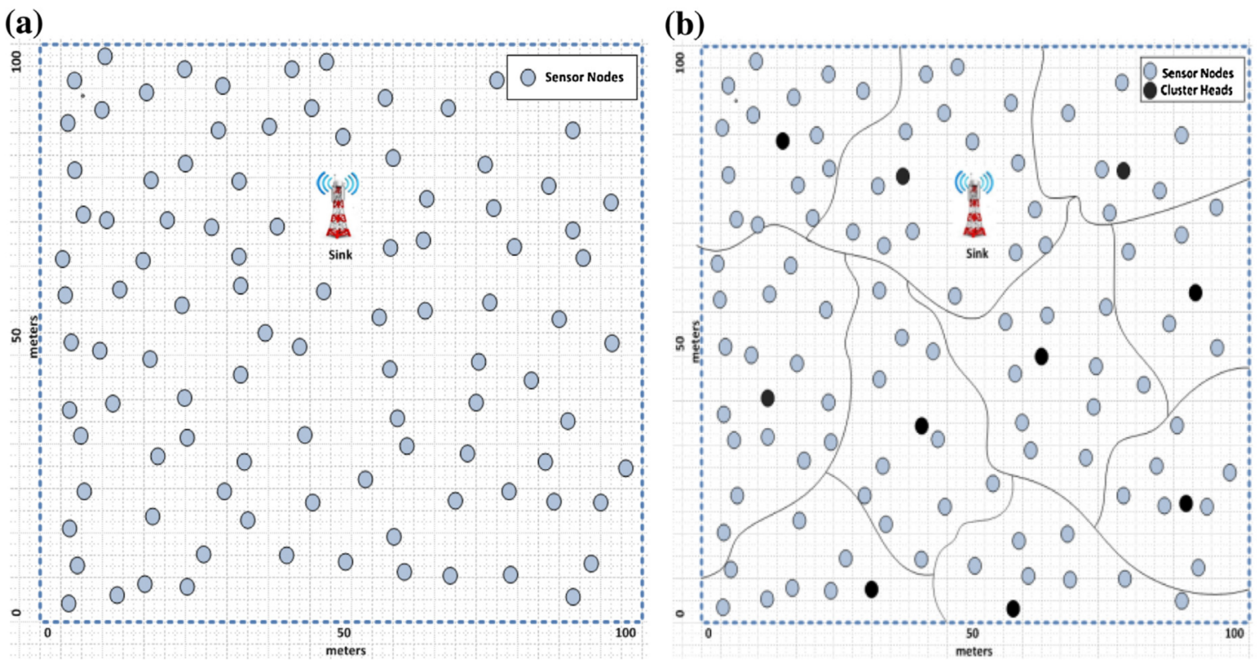


Figure 2.9: (a) Random deployment of SNs in a WSN field (b) Clustering formation in WSN field [6].

---

to have an efficient utilization of the node's energy, making it a central point when designing clustering protocols. One of the initial protocols that aimed on extending the lifetime of the network, by being energy efficient, is the LEACH [30] protocol. LEACH dynamically elects the CH in a round-robin fashion that allows each node in the cluster to become CH in varying rounds. The CHs collect the data from the other nodes, do aggregation and forward them to the BS. This approach increases the network lifetime, in comparison with direct communication, by balancing the energy consumption of each node.

Younis and Fahmy proposed the HEED [31] protocol, which is based on two clustering parameters: residual energy and node degree. This parametric combination leads to a load balancing feature and a more energy efficient protocol. Even though HEED was one of the prominent clustering protocols in the field, it suffered a few drawbacks, such as additional broadcast packet overhead caused by the CH selection procedures, hot-spots problems due to the more workload on CHs, specially the ones near the BS [32], and extraneous CH formation due to uncovered nodes [33].

Approaches were made based on the Fuzzy Logic, a Computational Intelligence technique that can be used in applications where there are uncertainties [34]. In [35] the authors presented a fuzzy-based protocol RE-TOPSIS that uses LEACH logic for the first CH election, and then the fuzzy logic on the following elections based on six factors:

1. Residual energy.
2. Distance between adjacency nodes;
3. Energy utilization rates;
4. Availability of neighboring nodes;
5. Distances between the nodes and the CHs;
6. Reliability index for completely devising the new scheme.

The CH election is made by first discovering the neighbors in the network, then each node calculates the rank of their neighbors and itself based on the previous factors. The neighbor with the highest rank announces itself in the network as a CH through broadcast. Then, the neighbors answer with *request joints* to the CH. After a threshold value, a new CH election takes place.

Many optimizations of the previously presented protocols have been developed over the years, but most of them tend to focus on the energy consumption and centrality, not taking much into account the quality of the connection between the nodes [29] [31]. The majority fail to be robust in scenarios with high mobility and require high processing due to their complexity [36], or have a considerable network overhead and do not consider mobile scenarios as [37]. Our proposal is a protocol of Cluster Head/Gateway election on a dynamic cluster, done on a passive way with low overhead, and combining multiple metrics that are relevant for aquatic monitoring environments in scenarios with high mobility.

2.4 Network Simulators

Testing new protocols and systems directly in the real-world is not always feasible and affordable. Besides, being able to test theories before applying them on real scenarios prevents a possible waste of resources. In order to minimize the gap between reality and simulation, the simulation mechanisms must be as accurate as possible.

2.4.1 UUV Simulator

Magalhães et al. [38] present the Unnamed Underwater Vehicles Simulator (UUV Simulator). It is a gazebo-based project with several algorithms for UUVs that implements design models, world models, sensor models, and control algorithms. This simulator allows hydrodynamic constrains, multi-robot systems, and the integration of new modules. There is an extensive Wikipedia on all the modules and tutorials to integrate new robots, worlds and other control algorithms. Being Gazebo based on ROS, it makes it easy to replicate in a Raspberry-Pi with sensors, which facilitates the translation from simulated code to real scenario.

Daniela et al. [3], based on this simulator's package, developed a simulator using ROS and Gazebo with a network module for USVs. Figure 2.10 shows the UUVs simulator structure overview. It was also implemented the DTN mOVE and the PAmuLQE routing protocol. Unfortunately, this simulator was focused only for a cluster/team of USVs. Therefore, in situations where it is required a large and disperse number of USVs, such as the simulation of several clusters, this simulator takes too much process power, limiting the number of USVs we can simulate on the provided hardware.

2.4.2 The ONE

The Opportunistic Protocol Simulator[2] (ONE) [39] is an open source free simulator written in Java with credibility within the community of researchers in DTNs. It contains a wide variety of models that continues to grow. The ONE allows the generation of node movements using different models, the reproduction of message traffic and routing, cache handling and the visualization of both mobility and message passing through its graphical user interface, general statistics, and others. As advantages, it is very easy to use and has a solid user community. However, once it is written in Java, ONE does not perform well for large simulations. It does not offer any link technology models nor radio propagation models.

[2]https://akeranen.github.io/the-one/

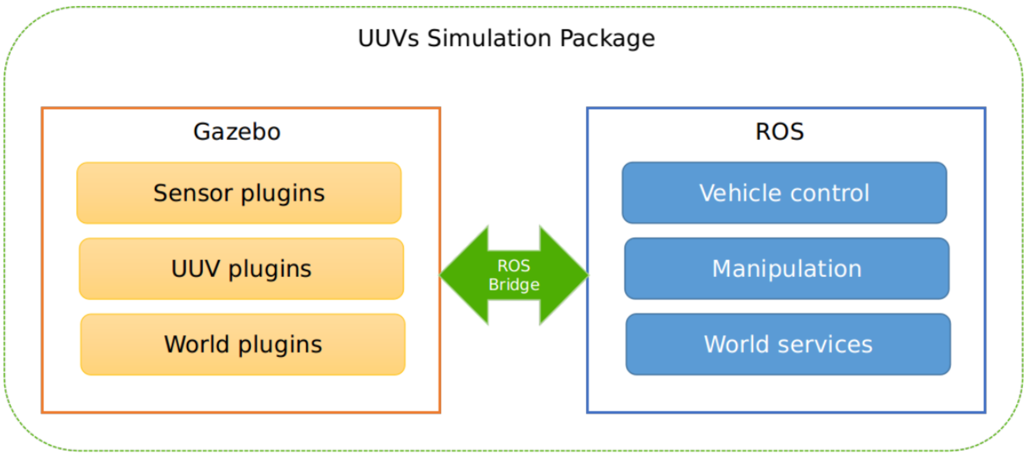


Figure 2.10: UUV Simulator-Software structure taken from Daniela et al. [3].

2.4.3 ns-3

Network Simulator 3 (ns-3) [3] is a discrete-event network simulator for Internet systems, targeted primarily for research and educational use. The simulation scenarios can be created using C++ or Python languages and they are run on command line with no GUI, even though there are tools, like *NetAnim*, that allow node mobility visualization during or after the simulation. It is not compatible with its previous predecessor ns-2, as it was completely rewritten [40]. ns-3 is licensed under the GPLv2 open source license.

As ns-3 is a general purpose network simulator, it needs to be configured with proper protocols at all levels to be used as a DTN simulator. Some models are data propagation protocol, mobility, traffic, link technology, and others.

As advantages, the communication models have been designed to easily match the interfaces to the standard Linux APIs. For this reason, it is possible to easily move a simulation setup to the real world and vice-versa. It has the capability to interact with existing real-world works using virtual TAP (layer 2) devices, connecting simulated and real nodes.

However, it lacks a user-friendly interface with debugging and visualization options. Its structure is rather complex and not easy to parameterize or change, even for more experienced users.

2.4.4 OMNeT++

OMNeT++ [4] is an extensible, modular, component-based C++ simulation library and framework, primarily for building network simulators. Although not being a network simulator itself, it has gained widespread popularity as a network simulation platform in the scientific community, as well as in industrial settings, and building up a large user community. It is an object-oriented, general purpose discrete event simulator. Components (modules) are programmed in C++, then assembled into larger components and models using a high-level language (NED). OMNeT++ simulation models are parameterized and configured for execution using configuration files with the *.ini* extension.

The network is simulated by the INET framework. However, it is not very well suited directly for DTNs or opportunistic simulations, as it focuses on IP-based networks and their protocol

[3]https://www.nsnam.org
[4]https://omnetpp.org

stack, which is overly-complex for opportunistic networks [41]. Even though, INET has useful mobility modules, like *BonnMotionMobility*, than can get from a plain text file the nodes position overtime, as each line describes the motion of one host .

Several other frameworks have been developed, but many are only compatible with earlier versions of OMNeT++.

OMNeT++ has a very sophisticated user-friendly interface, with many possibilities for visualizing and inspecting the simulated scenarios, with support for 2D and 3D graphics. It is highly modular, while clearly separating the node behaviour from the node parameters, making it very easy to run large parameter studies. It runs on all major operating systems, has a very maintained documentation, including user guides and tutorials, a wiki page, and a large community.

However, it does not offer the possibility of directly transferring simulation models to real implementations, e.g. directly to Linux distributions.

OMNeT++ presentes a very good performance on several tests for network simulations [42], and with its very user-friendly interface and structure, it is a choice to consider when deciding to build a simulator for DTNs.

Some projects on DTN/opportunistic networks, built upon OMNeT++, have been developed over the years, but many do not share their source code, are made upon an old version of OMNeT++, being now outdated and have an architecture too complex to update for low experienced OMNeT++ users, as is the case of the SAORS simulator [43], or are designed for spacial scenarios, like the DtnSim [5]. A recent and open-source project for opportunistic network simulations is presented next.

2.4.4.1 Opportunistic Protocol Simulator

The Opportunistic Protocol Simulator (OPS) [6] is a set of simulation models in OMNeT++ to simulate opportunistic networks developed by Udugama et al. [44]. There is also a scaled-down, light-weight version of OPS, the OPSLite [7], that is included on the official OMNeT++ Simulation Models and Tools list [8]. OPS has a modular architecture where different relevant protocols for opportunistic networks can be developed and plugged in. Figure 2.11 shows the OPS node architecture that consists of 5 layers.

Each of the layers can be configured through their parameters to behave as required.

1. Application Layer: consists of the *KHeraldApp* application that classifies data items as liked and non-liked and injects them uniformly over the simulation period.
2. Opportunistic Networking Layer: consists of the *KEpidemicRoutingLayer* which is used to forward data in an opportunistic network, based on the *Epidemic* strategy, and is capable of storing data in cache.
3. Link Adaptation Layer: is a simple pass-through layer (*KLinkAdaptLayer*) aimed to be extended in the future.
4. Link Layer: consists of the *KWirelessInterface* model that performs simple wireless communications.
5. Mobility: implements the movements of the mobile nodes in the scenarios. It can use any of the mobility models available in the INET4 Framework. In OPSLite, *omnetpp.ini* is configured to use BonnMotion (*BonnMotionMobility*) mobility model. Some sample traces with SLAW mobility (*trace-slaw-01.movements*, ...) are available in the *simulations* folder.

[5]https://bitbucket.org/lcd-unc-ar/dtnsim/src/master/

[6]https://github.com/ComNets-Bremen/OPS

[7]https://github.com/ComNets-Bremen/OPSLite

[8]https://omnetpp.org/download/models-and-tools

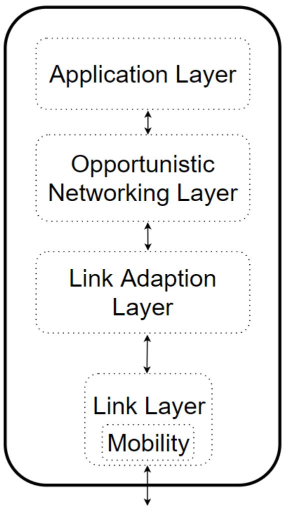


Figure 2.11: OPS Node Architecture.

This simulator is configured for a set of network-level results from the several layers to be collected in every simulation run.

OPS is a very simple and efficient simulator, being able to run a very high number of nodes within a simulation, but it does not have a radio model for aquatic scenarios, and its simple node architecture lacks some complexity that would make this node more robust for different scenarios. Beside these limitations, this project was an important source of knowledge and inspiration for the simulator proposed in this work.

2.4.5 Physical Layer

To achieve simulation results similar to real experiments, the simulated physical layer must represent the real scenario as accurate as possible. As this work aims on simulating USVs that move above water, the physical communication will have a different behaviour than a normal above ground open space communication. Above water communications deal with many reflections due to the water, which will spoil the communication quality within the network.

Daniela et al. [3] obtained some equations calculated from real world experiments using WiFi (2.4GHz 802.11n Wireless LAN) technology. These equations help modeling the physical layer of aquatic scenarios. SSI was modeled using results obtained with an antenna 15 cm above the drone level, resulting on a 4^{th} degree equation with a Mean Absolute Error (MAE) of 0.217 dBm, represented in Equation 2.6, and is valid for distances between 5 and 40 m (expressed by x).

$$SSI\_est(x) = -0.00002x^4 + 0.00051x^3 + 0.05897x^2 - 2.72277x - 57.56120, x \in [5, 40[. \quad (2.6)$$

In order to estimate the delay for each packet in the simulator, by dividing the number of bytes sent by the Achievable Throughput at a certain distance, the Achievable Throughput was modeled using results obtained with the antenna 15 cm above the drone level, resulting on a 4^{th} degree function with a MAE of 0.738 $Mbps$ valid for distances between 5 and 40 m (expressed by x), and it is presented in Equation 2.7.

$$AT(x) = -0.00018x^4 + 0.01603x^3 - 0.47286x^2 + 4.20788x + 14.6655, x \in [5, 40[. \quad (2.7)$$

Equation 2.8 represents the Link Stability between two USVs, influenced by the distance. This metric is used to estimate the packet loss in the simulator. For an antenna 15 cm above the drone level, it was obtained a 3^{rd} degree function with a MAE of 4.065% valid for distances between 5 and 40 m.

$$LinkStability = -0.002x^3 + 0.01048x^2 + 0.45476x + 97.2143, x \in [5, 40[. \tag{2.8}$$

This modeling of the real world physical communication will be used in the work of this thesis.

2.4.6 Energy Expenditure

This work uses wireless interfaces, namely WiFi technology, for the communication between the USVs. Table 2.2 presents some parameters obtained in practical tests found in [45], [46], [47], [48], [49] and [50]. It includes the energy consumption on three different wireless communication interfaces: Bluetooth, WiFi and WAVE. WAVE is the IEEE 802.11p standard for Wireless Access in Vehicular Environments (WAVE). From these parameters, we can observe that for their tested scenarios, the communication energy consumption of WiFi and WAVE is identical, even though WAVE has higher range.

Parameter	**Bluetooth**	**WiFi**	**WAVE**
interfaceType	BLE5	802.11n	802.11p
transmitSpeed	0.8 $Mbps$	80 $Mbps$	20 $Mbps$
transmitRange	70 m	300 m	900 m
scanInterval	1 s	0.25 s	0.25 s
scan Energy	0.6 mA	12 mA	12 mA
scanResponseEnergy	0.2 mA	9 mA	9 mA
transmitEnergy	12 mA	100 mA	100 mA
receiveEnergy	8 mA	66 mA	66 mA

Table 2.2: Parameters for the communication interfaces.

Finally, in order to keep a track of the energy spent by the communication [36] between the USVs during the simulations, and taking into account that during this work it will be used wireless communications, the following equations will be used to track the energy expenditure, based on a radio model similar to [7] with $\mathrm{E_{elec}} = 50nJ/bit$ as the energy dissipated by the radio to run the transmitter or receiver circuitry, and $\varepsilon_{\mathrm{amp}} = 100\ pJ/bit/m^2$ as the energy dissipation of the transmission amplifier. The energy expended during transmission and reception for a k bit message to a distance d between the transmitter and the receiver node is given by [51]:

$$E_{Tx}(k, d) = E_{elec} \times k + \varepsilon_{amp} \times k \times d^{\lambda} \tag{2.9}$$

$$E_{Rx}(k) = E_{elec} \times k \tag{2.10}$$

where λ is the path loss exponent and $\lambda \geq 2$.

2.5 Chapter Considerations

This chapter provided the fundamental knowledge to understand the remaining report. An overview on the relevant Link Quality Estimators for the connection between USVs, the behavior of a DTN architecture, namely the mOVE implementation, and several forwarding strategies for DTN were described. Also, some state of the art regarding cluster head election was presented. Several simulation tools were compared, where OMNeT++ was described with more detail as it will be used throughout this dissertation.

Chapter 3

A DTN Simulator for Swarms of Drones

Chapter 2 discussed the unavailability of simulation platforms for aquatic environments with an implemented DTN architecture, designed to simulate several swarms of drones. Therefore, the first phase of this dissertation is the development of a simulator. This chapter presents, in detail, the simulator development process, explaining how it was conceived and how it works. Section 3.1 describes the process behind the development of the AquaticDTNS. Section 3.2 describes the architecture, the flow of the messages within the architecture and between nodes, and the configuration of the OMNeT++ based simulator. Section 3.3 depicts the chapter summary and considerations.

Starting with real-world tests of the different network protocols and systems developed for aquatic monitoring USVs would not be an affordable task, specially when working in scenarios that involve a high number of USVs. It would be monetarily expensive, and also a waste of resources. In this way, it is essential to have a simulator that can approximate real-world scenarios as accurate as possible.

3.1 Developing a Simulator

In a previous work by Daniela Sousa [3], an emulator based on ROS and Gazebo, along with Magalhães et al. [38] UUV Simulator packages, was developed. On a first approach, this simulator was tested, but turned out to be too heavy for the available hardware (a laptop with intel i5 8th generation CPU and GTX1050m GPU). In situations with more than 10 USVs, the simulation would slow down until it collapsed.

As this dissertation aims to test scenarios involving clusters, i.e., several groups of USVs, a different simulator had to be used. As described in Section 2.4, to the best of our knowledge, there are no aquatic surface simulators with a DTN architecture developed and with a source code provided for other researchers to use. From that, we concluded that OMNeT++ was the best tool to develop a simulator. Some simulators based on OMNeT++ that shared their source code were tested. First, the simulator SAORS [43] was tested, but it showed several incompatibilities, manly because it was not updated since 2014. Another tested simulator was the OPSLite simulator developed by Udugama et al. [44]. This simulator was not designed for USVs simulation, so it presented a few setbacks of robustness, and it lacks a radio model based on aquatic surface communications. However, as it presented a quite simple architecture and a fair documentation, it was an important source of knowledge to help us understand how the OMNeT++ operates and how to project a DTN simulator using its library and framework.

3.2 Aquatic Surface Delay Tolerant Networks Simulator (AquaticDTNS)

The Aquatic Surface Delay Tolerant Networks Simulator (AquaticDTNS) is a set of simulation modules in OMNeT++ whose purpose is to simulate delay tolerant networks (DTN) in aquatic environments. It has a modular architecture where different protocols relevant to DTNs can be developed and tested. Its architecture was inspired by the OPS simulator [44], one of the few recent opportunistic simulators developed using the OMNeT++ framework.

3.2.1 Architecture

AquaticDTNS is built upon the OMNeT++ framework, so a fair understanding of the basic concepts of OMNeT is suggested but not required for basic usage, as it is very straight-forward. OMNeT's basic concepts are modules and messages. A simple module is the basic unit of execution that accepts messages from other modules or itself. Each module has a *handle* method that, according to these messages, executes a piece of code.

The simulator architecture is a protocol stack of protocol layers, representing the OMNeT++ modules being used, that constitute the architecture of a DTN node. As represented in Figure 3.1, each layer focuses on a specific area of operation of a DTN node.

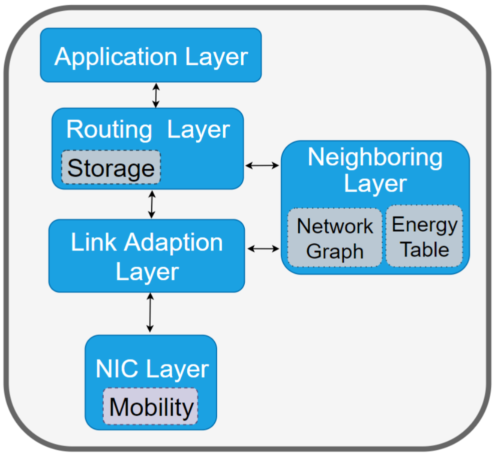


Figure 3.1: Node Architecture.

3.2.1.1 Application Layer

The Application Layer consists of the application module where data from sensors of each node is generated, and then sent to the layers below to be forwarded to its destination. We can configure several proprieties on this node, being the main ones:

- Data generation duration;
- Data generation periodicity;
- Data size;
- Data expiration time;
- Data destination address;

- Which nodes will generate data;
- Generate a chosen number of messages right at the beginning of the simulation;
- Data's maximum number of hops.

3.2.1.2 Routing Layer

The Routing Layer consists on the Routing module where the user can easily implement and test different DTN protocols and data propagation mechanisms. For example, for this dissertation it will implement different forwarding strategies and gateway election mechanisms. It is also within this layer that we can access the Storage, where the DTN routing protocol shall save data messages. We can configure several properties on this node, being the main ones:

- The Routing protocol we want to test;
- The maximum storage size it can use;
- Packet's maximum number of hops;
- Starting gateway;
- Gateway list;
- Gateway check period.

3.2.1.3 Neighboring Layer

The Neighboring Layer consists of the neighboring module fundamental for the correct operation of the network. It is responsible for the neighboring discovery process. This module is responsible for sending beacons (*neighbor announcements*) periodically, which contain summary information of the node as its position, neighborhood graph and energy table. When a beacon packet is received, the module updates the graph and the energy table of the neighbors with the received information. It has several methods to compute the link quality between direct neighbors, and to calculate the best path between the transmitter and the gateway. This module is also responsible for updating the node's table of direct neighbors by erasing and adding new ones. We can configure several proprieties on this node, being the main ones:

- Periodicity of beacons;
- Time before start sending beacons;
- Preferred Network Interface;
- Maximum size of the graph and energy table;
- Amount of energy at the beginning of the simulations of the Network Interface Controller's (NIC) dedicated battery.

3.2.1.4 Link Adaptation Layer

The Link Adaptation Layer is responsible for forwarding the packets received by the NICs to the respective upper layers according to their type, and also to forward internal communication between the NICs and the Neighboring Layer. Beacons and Data Requests are forwarded to the Neighboring Layer, while Data Messages and Acknowledgements are forwarded to the Routing Layer. Figure 3.2 presents the remaining procedures that are provided when an external packet is received. As this simulator has a modular architecture, mechanisms that convert messages from one form to another, as tunneling of packets, can easily be implemented. Currently, this layer does not require additional configuration.

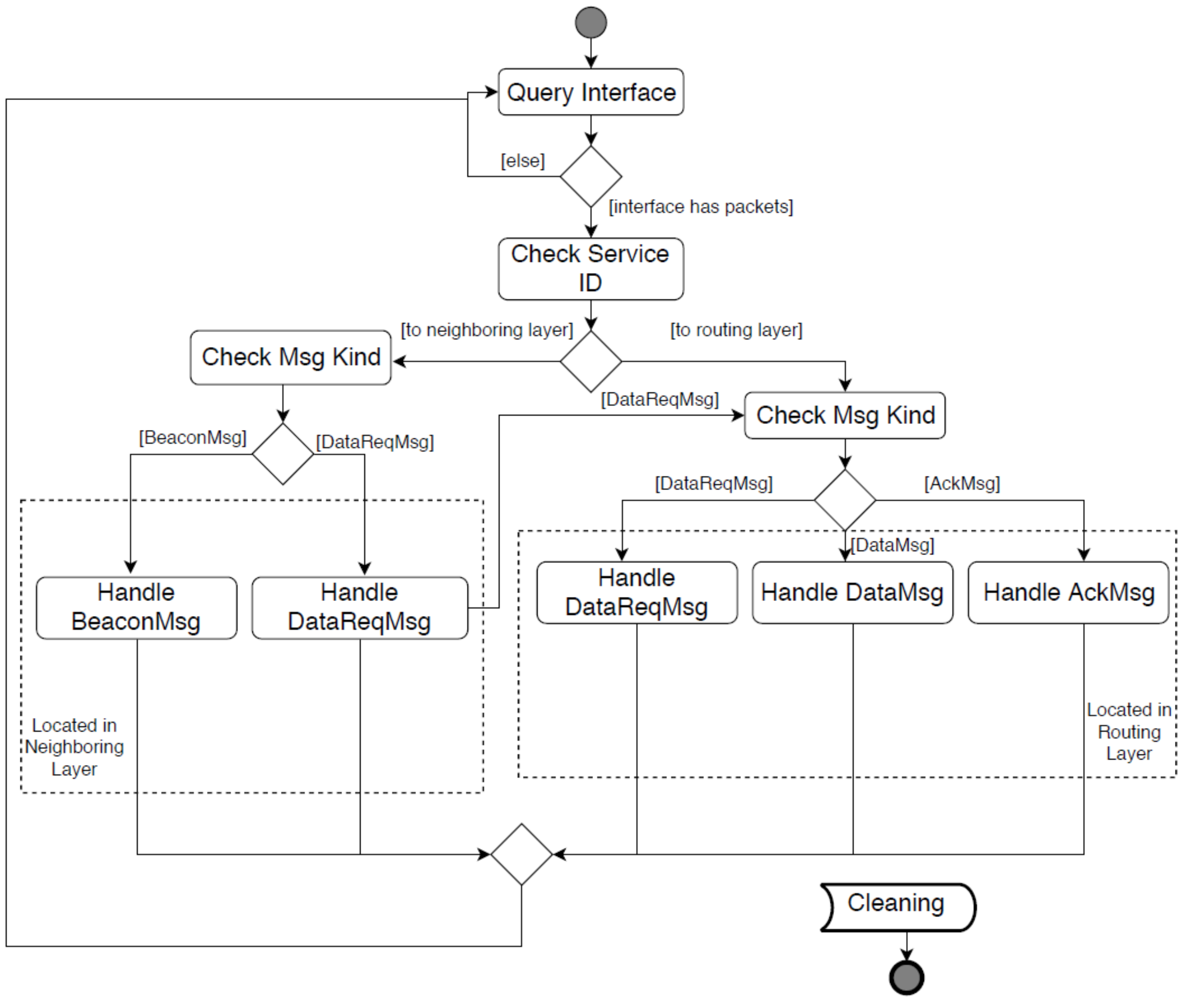


Figure 3.2: AquaticDTNS external packet reception process flowchart.

3.2.1.5 NIC Layer

The NIC is where the Network Interface modules are. We can configure the network interface modules according to the proprieties of the ones we chose. We have implemented a wireless interface (for Bluetooth and WiFi) and used the previously mentioned equations ((2.6), (2.7) and (2.8)), that simulate the behaviour of the wireless on above water communications. As these equations depend only on the distance between two nodes, here it is also implemented the Mobility module provided by the INET Framework that returns the position of the node. The mobility module from the INET Framework also handles the node mobility throughout the simulation. Some of the wireless module configuration proprieties are:

- Range;
- Bandwidth Bitrate;
- Header Size;
- Minimum RSSI;
- Neighbor scan interval.

And for the mobility module we have:

- Mobility type;
- Trace file;
- If mobility is 3D.

3.2.2 Message Flow

In OMNeT++ the modules and nodes communicate through messages. We divided the messages in two types: internal messages, that are messages that only flow within the node; and external messages, messages sent to the outside or arrive from the outside, and then are treated and passed along the node's layers.

In Figure 3.3 we can see the flow of the messages described in the next paragraphs.

Internal Messages

Internal messages can be sent to other modules within the node to request or share information, or can be sent to the module itself, in order to schedule an event.

- **NeighboringListMsg**: This message comes from the network interface and contains the number and table of direct neighbors.
- **GraphUpdtMsg**: This message is used to pass and receive information regarding the neighborhood graph. When the NIC detects that there are no neighbors, it informs the Neighboring module. The Neighboring module uses this message to pass the neighborhood graph to the Routing module for forwarding decisions.
- **PcktIsSentMsg**: This message is sent from the NIC module to the Neighboring module every time a packet is sent or received with its size and sending or reception time.
- **EnerTableMsg**: This message is used by the Neighboring module to send to the Routing module the Neighbors' Energy table, along with a time stamp of when this information was processed.
- **BeaconInfoMsg**: The *BeaconInfoMsg* is a copy of the external *BeaconMsg* that is processed first by the Neighboring module, updated and sent to the Routing Layer for forwarding decisions.

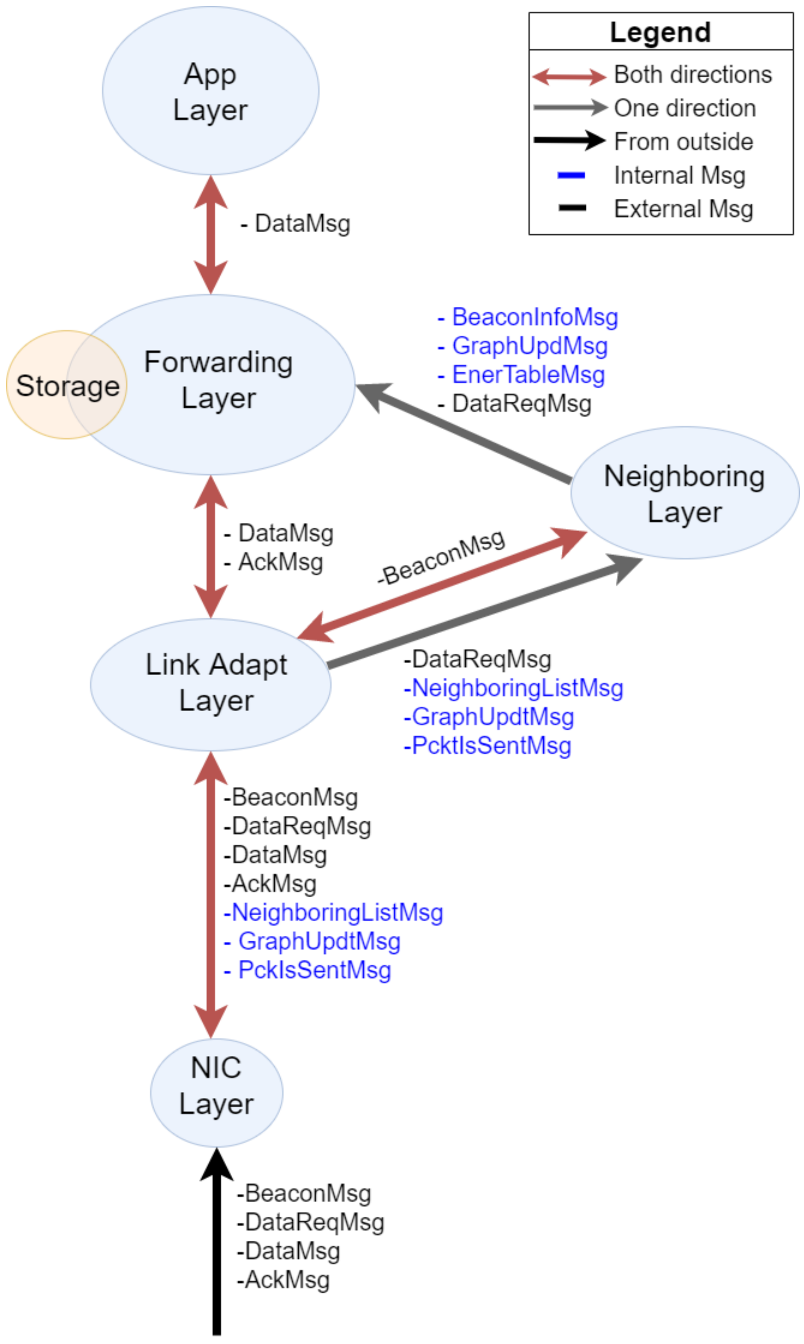
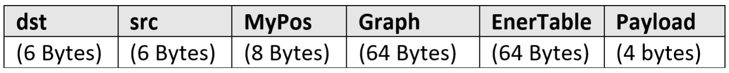


Figure 3.3: Message flow in AquaticDTNS.

External Messages

External messages are the ones exchanged between the nodes. This transaction is emulated by the NIC module that sends data from a node directly to the other, applying the physical influences of the simulated environment. These messages are the following:

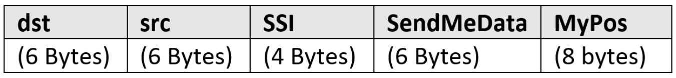
- **BeaconMsg**: The *BeaconMsg* is the also called *Neighboring Announcement* packet that is broadcasted periodically into the network. It contains the coordinates of the sender's position, neighboring graph and energy table. This packet is represented in Figure 3.4.



dst	src	MyPos	Graph	EnerTable	Payload
(6 Bytes)	(6 Bytes)	(8 Bytes)	(64 Bytes)	(64 Bytes)	(4 bytes)

Figure 3.4: BeaconMsg packet.

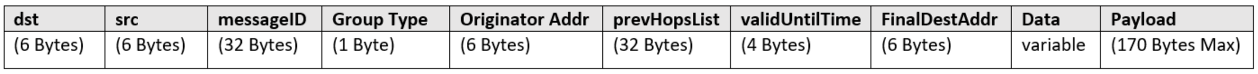
- **DataReqMsg**: This message represents the packet sent by the Neighbors as an answer to the received *BeaconMsg*. It contains the calculated SSI of the connection between the two nodes and a flag confirming its readiness to received any available data. This packet's goal is to establish a connection between the two nodes and trigger the transmission of stored data messages, if so decided by the Routing Layer. This packet is represented in Figure 3.5.



dst	src	SSI	SendMeData	MyPos
(6 Bytes)	(6 Bytes)	(4 Bytes)	(6 Bytes)	(8 bytes)

Figure 3.5: DataReqMsg packet.

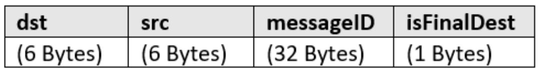
- **DataMsg**: This message contains the data generated from the sensors of the USV. The data is identified by its *messageID* and classified by its group type. This packet also contains the address of the node that generated the data, a life time for the packet, and a list of previous hops, if it is a copy being forward, so it tracks how many hops it goes by from the source to its final destination (gateway). This packet is represented in Figure 3.6.



dst	src	messageID	Group Type	Originator Addr	prevHopsList	validUntilTime	FinalDestAddr	Data	Payload
(6 Bytes)	(6 Bytes)	(32 Bytes)	(1 Byte)	(6 Bytes)	(32 Bytes)	(4 Bytes)	(6 Bytes)	variable	(170 Bytes Max)

Figure 3.6: DataMsg packet.

- **AckMsg**: This message confirms the reception of a *DataMsg* by providing the packet's received ID and a flag indicating if this packet reached its final destination. This packet is represented in Figure 3.7.



dst	src	messageID	isFinalDest
(6 Bytes)	(6 Bytes)	(32 Bytes)	(1 Bytes)

Figure 3.7: AckMsg packet.

3.2.3 Configuration and Operation Overview

In Figure 3.8 we can visualize the AquaticDTNS architecture and operation flowchart. The process of configuration starts by defining the network's *NED* file. *NED* language is used by the OMNeT++ to describe a module, so each module programmed in C++ in OMNeT++ also has a *NED* [1] file associated to describe it. In the network *NED* file, one can define the type of nodes to be used during the simulation. In the case of this dissertation, we are using a node with the previously presented DTN architecture, as represented in Figure 3.9. This simulator uses the sublayer MAC of OSI Layer 2 on each node to provide addressing and channel access control mechanisms.

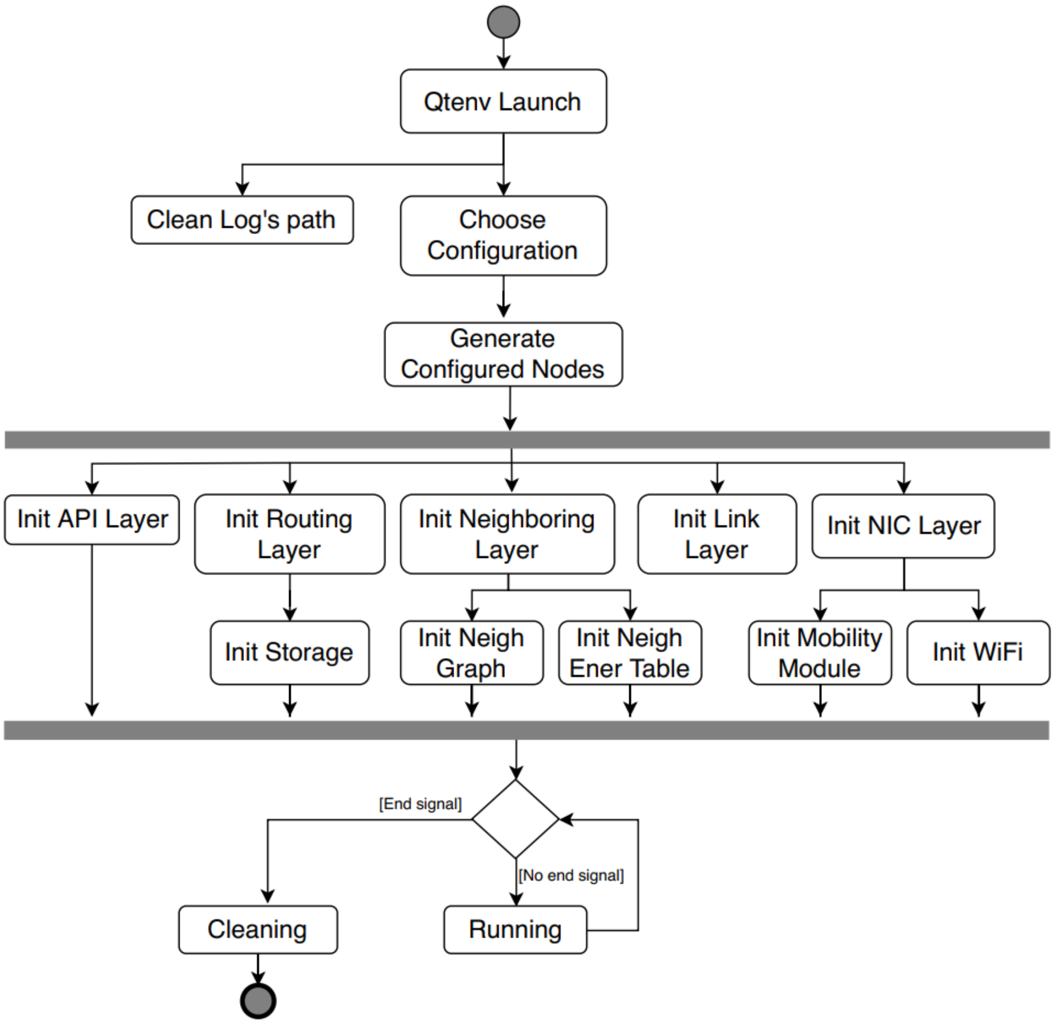


Figure 3.8: AquaticDTNS operation flowchart.

The next step is the configuration of an *INI* [2] OMNeT++ file with the configuration of the modules. In this file we define a set of macro variables that will be responsible for setting the main characteristics of the data collection uploading, simulation, and statistical analysis procedure. The main description is done as follows:

- **General parameters**:
  - *network*: the name of the *NED* file configured for the network that specifies the type of nodes in the simulation;
  - *sim-time-limit* and *repeat*: sets the maximum duration of the simulation in seconds and how many times it repeats itself;
  - *numNodes*: here we can set the total number of nodes or a set of options of the number of nodes that can be selected before running the simulation on the *Qtenv* graphical mode.

[1]https://doc.omnetpp.org/omnetpp/manual/
[2]https://doc.omnetpp.org/omnetpp/IDE-Overview.pdf

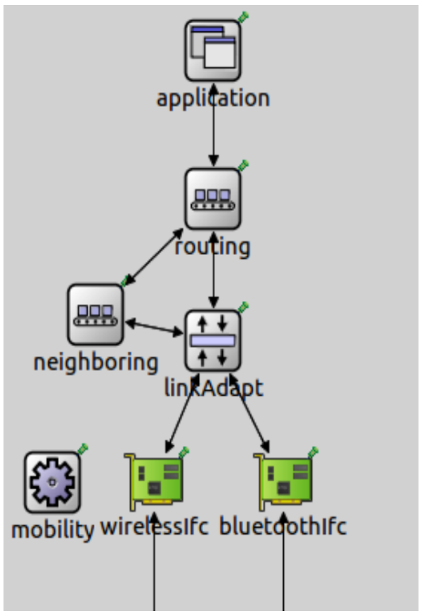


Figure 3.9: Example of AquaticDTNS node configuration in OMNeT++.

- **Application module parameters**:
  - *dataGenerationInterval*, *endGeneratingMsg* and *ttl*: the former sets the period of time a data message is generated, the latter sets the simulation time for the end of data message generation, and the third (time to leave) sets the expiration time of each data message;
  - *dataSizeInBytes*: defines the size of the generated data;
  - *destinationAddr*: defines the destination node by its MAC address;
  - *ThatGen* and *nodesGenMsg*: the former is a flag that decides if the latter parameter is a string with the list of nodes generating messages during the simulation;
  - *startMultipleMsg* and *numMultipleMsg*: the former is a flag that indicates if the application module should generate several messages right at the start of the simulation, and the latter defines how many messages are generated at the beginning of the simulation, if the previous parameter is true;
  - *hopsListSize*: sets the number of maximum hops the generated data message can travel through.
- **Routing module parameters**:

  The following parameters are not mandatory and can be used, depending on the protocol/strategy to be used:
  - *routingLayer*: here we define the routing/forwarding protocol;
  - *maximumCacheSize*: defines the maximum size of the DTN storage during the simulation;
  - *maximumHopCount*, *useTTL* and *max_age*: the former sets the maximum number of hops for a data message to be stored, and the latter (time to leave) is a flag that, if active, checks every time a data message is received or pulled out from storage. If the stored message has reached its expiration time, this message is deleted from the

storage. The third parameter can be used to extend the time a message is kept alive in the storage;

- *waitBFsend*: can be used to set a time for the routing to wait between finishing processing a received message and the start of the process of sending another message. This time can be used to give time for the storage to update itself after being used on the reception of a data message before consulting it for new data messages to be sent;
- *gateway_ list*, *actual_ gateway* and *gwCheckPeriod*: the former can be used to provide a list of gateways, the second can be used to define a gateway at the beginning of the simulation, and the latter parameter sets the period of time a routing protocol can use to recheck or recalculate its gateway.

• **Neighboring module parameters**:

- *N_ nodes*: sets the maximum number of nodes it can store information about;
- *EnergyStart*: sets the energy in *Jules* dedicated to a node's network module;
- *BS_ position*: sets the coordinates in land of the Base Station (BS);
- *sendBeaconInterval* and *maximumRandomBackOffDuration*: the former sets the period a node sends the *BeaconMsg*, and the latter sets the time it can wait before starting to send the *BeaconMsgs*;
- *WifiFirst*: Flag that confirms if the main NIC is WiFi;
- *max_ absent_ time*: is the maximum time this module waits for new information about a neighbor's presence before deleting it from its table;
- *resetGPeriod*: this parameter can be used to force a periodically cleaning of the neighbors' graph.

The next two models are within the NIC Layer:

• **Wireless module and/or Bluetooth module parameters**:

- *wirelessRange* and *minSSI*: the first parameter defines the maximum wireless range of this node. The *minSSI* is the minimum RSSI calculated between the two nodes, under which it considers the neighbor out of range for communication;
- *neighborScanInterval* and *limitQueueTime*: the former is the period of time the NIC searches for direct neighbors and updates its table. The latter is the maximum time a packet can be in queue to be sent;
- *bandwidthBitRate* and *wirelessHeaderSize*: the former parameter sets the bandwidth bit rate of the NIC, and the latter parameter defines the header size (which in wireless interfaces differs, for example, from WiFi to Bluetooth).

• **Mobility module parameters**:

Depending on the mobility type being used, some other configurations might be needed. If using INET's framework, a check up on INET's API [3] is advised:

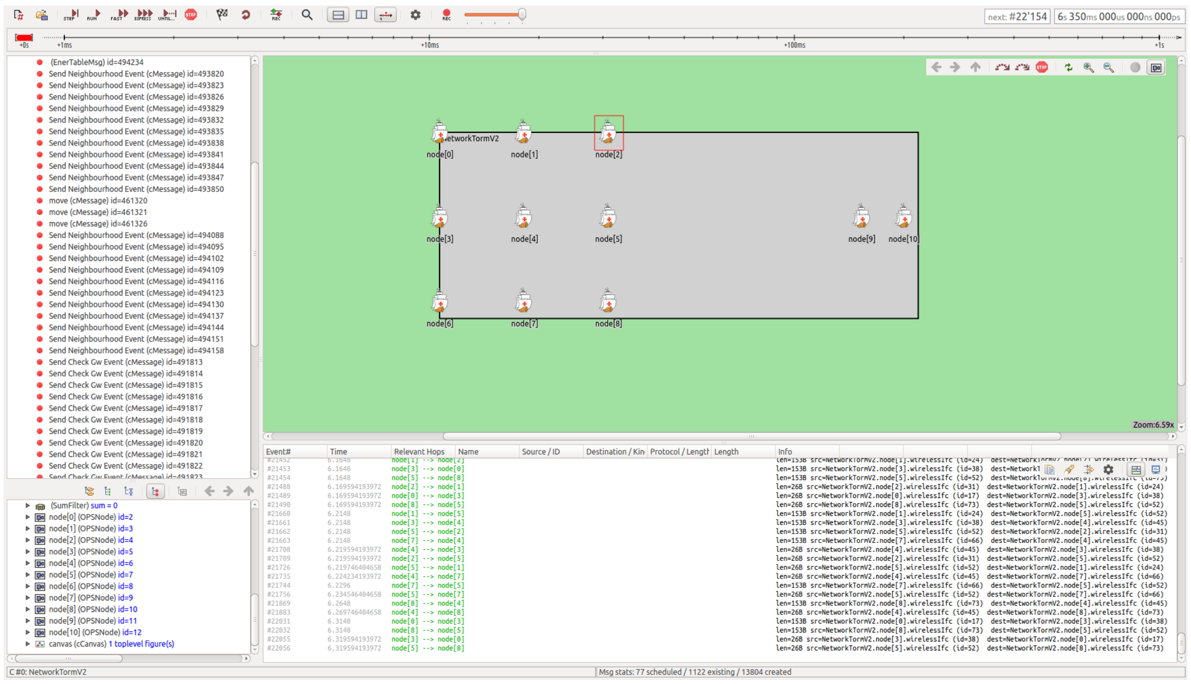
- *mobilityType*: here we define the type of mobility of the nodes throughout the simulation. We use the INET's *BonnMotionMobility* [4] that reads the node's position during simulation from a file, where every line describes the motion of one host/node;

[3]https://doc.omnetpp.org/inet/api-current/neddoc/index.html
[4]https://doc.omnetpp.org/inet/api-current/neddoc/inet.mobility.single.BonnMotionMobility.html

- *traceFile*: is the file that contains the node's position during the simulation to be processed by the mobility module;
- *is3D*: indicates whether the trace file contains triplets or quadruples parameters for the time and coordinates.

3.2.4 Qtenv Graphical Runtime Environment

After the configuration of the *INI* file, when running a simulation, a Qtenv graphical interface appears where we can chose the configuration set up, and other parameters if required by the configuration, and also visualize the simulation during its course. Qtenv helps to get a detailed picture of the state and history of the simulation at any point of its execution, as shown in Figure 3.10.



```
(EnerTableMsg) id=494234
Send Neighbourhood Event (cMessage) id=493820
Send Neighbourhood Event (cMessage) id=493823
Send Neighbourhood Event (cMessage) id=493826
Send Neighbourhood Event (cMessage) id=493829
Send Neighbourhood Event (cMessage) id=493832
Send Neighbourhood Event (cMessage) id=493835
Send Neighbourhood Event (cMessage) id=493838
Send Neighbourhood Event (cMessage) id=493841
Send Neighbourhood Event (cMessage) id=493844
Send Neighbourhood Event (cMessage) id=493847
Send Neighbourhood Event (cMessage) id=493850
move (cMessage) id=461320
move (cMessage) id=461321
move (cMessage) id=461326
Send Neighbourhood Event (cMessage) id=494088
Send Neighbourhood Event (cMessage) id=494095
Send Neighbourhood Event (cMessage) id=494102
Send Neighbourhood Event (cMessage) id=494109
Send Neighbourhood Event (cMessage) id=494116
Send Neighbourhood Event (cMessage) id=494123
Send Neighbourhood Event (cMessage) id=494130
Send Neighbourhood Event (cMessage) id=494137
Send Neighbourhood Event (cMessage) id=494144
Send Neighbourhood Event (cMessage) id=494151
Send Neighbourhood Event (cMessage) id=494158
Send Check Gw Event (cMessage) id=491813
Send Check Gw Event (cMessage) id=491814
Send Check Gw Event (cMessage) id=491815
Send Check Gw Event (cMessage) id=491816
Send Check Gw Event (cMessage) id=491817
Send Check Gw Event (cMessage) id=491818
Send Check Gw Event (cMessage) id=491819
Send Check Gw Event (cMessage) id=491820
Send Check Gw Event (cMessage) id=491821
Send Check Gw Event (cMessage) id=491822
Send Check Gw Event (cMessage) id=491823
```

```
(SumFilter) sum = 0
node[0] (OPSNode) id=2
node[1] (OPSNode) id=3
node[2] (OPSNode) id=4
node[3] (OPSNode) id=5
node[4] (OPSNode) id=6
node[5] (OPSNode) id=7
node[6] (OPSNode) id=8
node[7] (OPSNode) id=9
node[8] (OPSNode) id=10
node[9] (OPSNode) id=11
node[10] (OPSNode) id=12
canvas (cCanvas) 1 toplevel figure(s)
```

```
Event#  Time            Relevant Hops        Name  Source / ID  Destination / Kin  Protocol / Lengtł  Length  Info
#21452  6.1648          node[1] --> node[2]     len=153B src=NetworkTormV2.node[1].wirelessIfc (id=24)  dest=NetworkTormV2.node[2].wirelessIfc (id=31)
#21453  6.1648          node[3] --> node[0]     len=153B src=NetworkTormV2.node[3].wirelessIfc (id=38)  dest=NetworkT
#21454  6.1648          node[5] --> node[8]     len=153B src=NetworkTormV2.node[5].wirelessIfc (id=52)  dest=NetworkTormV2.node[8].wirelessIfc (id=73)
#21488  6.169594193972  node[2] --> node[1]     len=26B src=NetworkTormV2.node[2].wirelessIfc (id=31)  dest=NetworkTormV2.node[1].wirelessIfc (id=24)
#21489  6.169594193972  node[0] --> node[3]     len=26B src=NetworkTormV2.node[0].wirelessIfc (id=17)  dest=NetworkTormV2.node[3].wirelessIfc (id=38)
#21490  6.169594193972  node[8] --> node[5]     len=26B src=NetworkTormV2.node[8].wirelessIfc (id=73)  dest=NetworkTormV2.node[5].wirelessIfc (id=52)
#21660  6.2148          node[1] --> node[5]     len=153B src=NetworkTormV2.node[1].wirelessIfc (id=24)  dest=NetworkTormV2.node[5].wirelessIfc (id=52)
#21661  6.2148          node[3] --> node[4]     len=153B src=NetworkTormV2.node[3].wirelessIfc (id=38)  dest=NetworkTormV2.node[4].wirelessIfc (id=45)
#21662  6.2148          node[5] --> node[2]     len=153B src=NetworkTormV2.node[5].wirelessIfc (id=52)  dest=NetworkTormV2.node[2].wirelessIfc (id=31)
#21663  6.2148          node[7] --> node[4]     len=153B src=NetworkTormV2.node[7].wirelessIfc (id=66)  dest=NetworkTormV2.node[4].wirelessIfc (id=45)
#21708  6.219594193972  node[4] --> node[3]     len=26B src=NetworkTormV2.node[4].wirelessIfc (id=45)  dest=NetworkTormV2.node[3].wirelessIfc (id=38)
#21709  6.219594193972  node[2] --> node[5]     len=26B src=NetworkTormV2.node[2].wirelessIfc (id=31)  dest=NetworkTormV2.node[5].wirelessIfc (id=52)
#21726  6.219746404658  node[5] --> node[1]     len=26B src=NetworkTormV2.node[5].wirelessIfc (id=52)  dest=NetworkTormV2.node[1].wirelessIfc (id=24)
#21735  6.224234193972  node[4] --> node[7]     len=26B src=NetworkTormV2.node[4].wirelessIfc (id=45)  dest=NetworkTormV2.node[7].wirelessIfc (id=66)
#21744  6.2296          node[7] --> node[5]     len=153B src=NetworkTormV2.node[7].wirelessIfc (id=66)  dest=NetworkTormV2.node[5].wirelessIfc (id=52)
#21756  6.234546404658  node[5] --> node[7]     len=26B src=NetworkTormV2.node[5].wirelessIfc (id=52)  dest=NetworkTormV2.node[7].wirelessIfc (id=66)
#21869  6.2648          node[8] --> node[4]     len=153B src=NetworkTormV2.node[8].wirelessIfc (id=73)  dest=NetworkTormV2.node[4].wirelessIfc (id=45)
#21883  6.269746404658  node[4] --> node[8]     len=26B src=NetworkTormV2.node[4].wirelessIfc (id=45)  dest=NetworkTormV2.node[8].wirelessIfc (id=73)
#22031  6.3148          node[0] --> node[3]     len=153B src=NetworkTormV2.node[0].wirelessIfc (id=17)  dest=NetworkTormV2.node[3].wirelessIfc (id=38)
#22032  6.3148          node[8] --> node[5]     len=153B src=NetworkTormV2.node[8].wirelessIfc (id=73)  dest=NetworkTormV2.node[5].wirelessIfc (id=52)
#22055  6.319594193972  node[3] --> node[0]     len=26B src=NetworkTormV2.node[3].wirelessIfc (id=38)  dest=NetworkTormV2.node[0].wirelessIfc (id=17)
#22056  6.319594193972  node[5] --> node[8]     len=26B src=NetworkTormV2.node[5].wirelessIfc (id=52)  dest=NetworkTormV2.node[8].wirelessIfc (id=73)
```

```
C #0: NetworkTormV2        Msg stats: 77 scheduled / 1122 existing / 13804 created
```

Figure 3.10: The main window of Qtenv.

3.3 Chapter Considerations

This chapter focused on the simulator developed throughout this dissertation. It started with the problem definition that led to the development of the simulator. We presented the proposed simulator starting with its architecture, where its layers were described. Then, the flow of the messages within the architecture of the node and also of the messages exchanged between the nodes was also presented. Lastly, the configuration of the simulator was presented, describing with some detail its parameters. On the next chapter, in Section 4.2, some forwarding strategies are presented whose simulation results will be compared with the results from Daniela et al. [3] in Section 5.2 in order to validate this simulator.

Chapter 4

Network Mechanisms

This chapter describes in detail the architecture and implementation of several mechanisms and strategies supporting the communication over DTN aquatic networks. Section 4.1 presents an overview of the aquatic platform architecture. Section 4.2 describes the behavioral flow of the proposed DTN forwarding schemes. Section 4.3 describes the behavioral flow of the proposed Gateway election strategies. Section 4.4 presents the chapter summary and final considerations.

4.1 Architecture Overview

USVs perform monitoring tasks working in groups, also denoted as clusters. As illustrated in Figure 4.1, within each cluster there is a CH, also referred as Gateway (GW). In this case, it is assumed that a GW has long range communication capability allowing the communication with the Base Station (BS) to transmit the environmental information. All nodes of the same cluster must have a path, even if by several hops, to the GW. In order to disseminate their data to the GW, an efficient forwarding protocol is necessary. The nodes also should be able to leave their cluster and join others, or even create a new cluster. When such happens, a new GW must be elected, so that the delivery of the clusters' data to the BS continues. Such behaviour also depends on a protocol that efficiently elects the GW.

4.2 Data Gathering

The USVs deployed on the aquatic environment will be gathering relevant sensory information from their monitoring operations. This gathered data must be forwarded to the BS. However, between the gathering of the data and its delivery to the BS, several processes will occur for an efficient delivery task. These processes, that include neighborhood classification, routing, and others, depend on several calculations, classifications and decisions.

4.2.1 Link Quality Estimators

The forwarding strategy implemented in this dissertation uses Link Quality Estimators (LQE) to chose the path for the transmission of each packet, according to the result of a previous evaluation process. The LQE calculation is based on the equations presented in Section 2.3.1 from [3]. However, it has some differences, as we do not use the bit rate computed with the observed delay on each received packet. Instead, we use the link stability between nodes.

In our work, this evaluation process classifies a mobile node according to the following information:

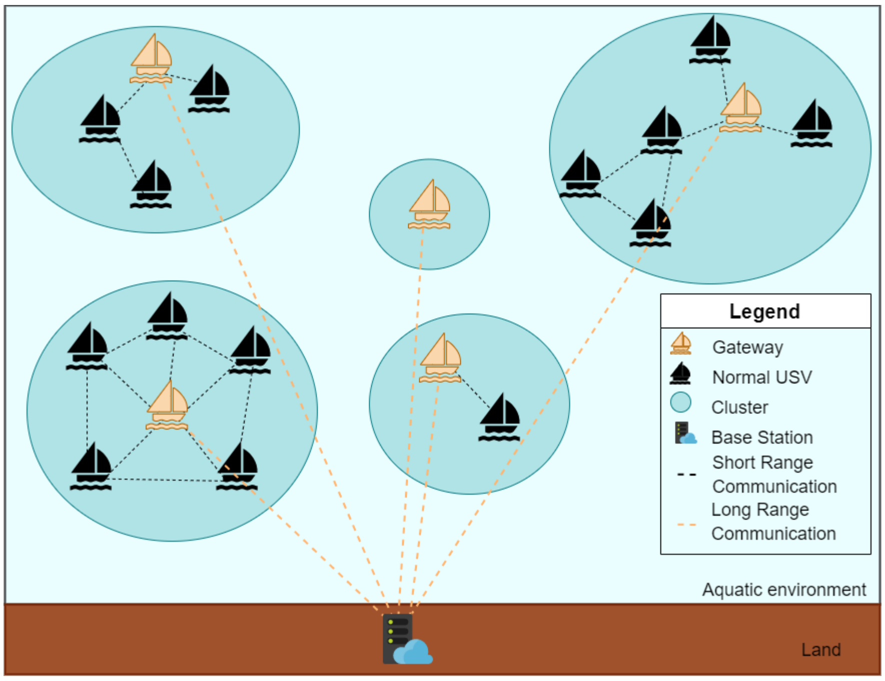


Figure 4.1: Aquatic-Wide Platform Overview.

- The estimated Signal Strength Indicator (SSI) between two nodes, from an estimation of the SSI given the distance between two USVs. By estimating the SSI, instead of using the hardware and WiFi packets to measure it, we save resources (processing time and energy), turning this into a passive quality estimator.
- The estimated link stability between the two nodes. Similar to the reasons of using the estimated SSI, the estimated link stability estimates the quantity of packets that are well received between two USVs according to the distance between them.

From the previous information, it is added to the adjacency graph a weight factor that derives directly from the LQE, and it is computed as follows [3]:

$$Weight = (1 - Link\_Quality) \times 100 \tag{4.1}$$

The *Link_ Quality* is given from the normalized values of the estimated SSI and Link Stability, differing from [3] as we do not use the bit rate computed with the observed delay on each received packet:

$$LinkQuality = (1 - \beta) \times SSI\_est\_norm + \beta \times Link\_stability\_norm \tag{4.2}$$

The factor β is a parameterizable value between 0 and 1 in the *Neighboring module*. It controls the weight of each of the two parameters, balancing their given relevance. In this work it is given the same weight to both parameters. The $Link\_Quality$ remains between 0 and 1.

4.2.2 Neighborhood Graph Update

Keeping the awareness of the surrounding nodes within the cluster for routing decisions leads to a more efficient utilization of the cluster's network. The neighborhood graph is a key part for the correct operation of the routing process, as many decisions will come based on the information provided by this graph. Each node will receive the network graph in every *beacon* transmitted by each node (USV), that is sent as a neighbor announcement. Then, the link quality between the computing node and its 1-hop neighbors is not updated, except for the path between the computing node and the node that sent the *beacon*, which is calculated and updated in every beacon reception. Of all the non-direct neighbors of the computing node, only the links that do not contain the computing node in the path are updated.

Figure 4.2 presents a practical example. Node D receives a packet from node B. When the packet is processed, the link quality between the two nodes is calculated and updated. However, the links D-C and D-E are not updated in the graph because nodes C and E are direct neighbors of node D, so the weight (that derives from the LQE, equation 4.1) of the links only changes when node D receives a packet directly from one of these nodes. The remaining links are only updated if a path between B and them exists without passing through node D. So, the weight of link E-F remains unchanged, the others are updated. Such behaviour happens because the node D can receive more recent information from nodes directly connected to this link.

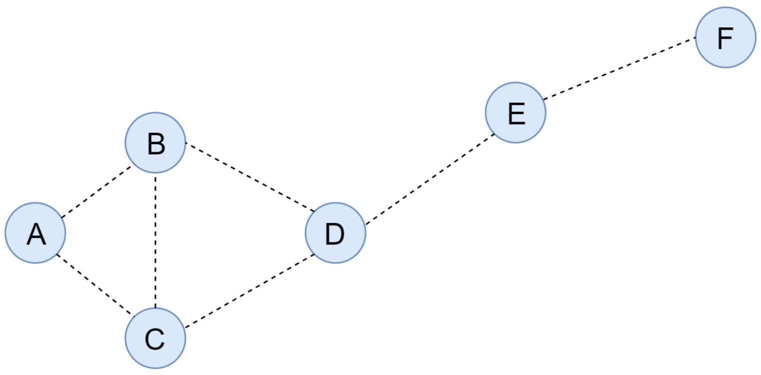


Figure 4.2: Network connection graph [3].

4.2.3 DTN Forwarding Strategies

On a DTN network it is important to have a routing protocol that can forward the stored data to its destination with efficiency and reliability, even if a node is mobile and there is no direct communication between the node and the GW. Therefore, the routing/forwarding strategy should present the lowest delay, introduce the least overhead on the network, but still maintain the highest delivery ratio.

4.2.3.1 Epidemic Strategy

The Epidemic strategy is a very popular flooding-based forwarding mechanism that aims to maximize the delivery ratio and minimize the E2E delivery rate, without having any concerns about the network resources. Considering a hypothetical scenario, where nodes would have infinite storage and CPU, and a null time to transmit messages between nodes, this routing protocol would have the highest delivery ration with the lowest delivery time. However, in real-world scenarios the available resources are limited, leading such mechanism to abuse of the network's resources, so this strategy will be used as basis for comparison.

4.2.3.2 Passive LQE based forwarding strategy

As described in section 2.3.2, the work in [3] proposed the Passive multi-hop Link Quality Estimator with neighbor acknowledgement (PAmuLQE-NACK), where the packets are forwarded accordingly to the best quality path from the source to the GW. A custody-transfer occurs, where each node keeps the packet until it receives the acknowledgement from the next node, deleting it afterwords. This way there is only one copy of the packet in the network by each time.

When implementing the previous protocol, the following situation was detected. The PAmuLQE-NACK had a Loop Avoidance mechanism, whose goal was to prevent data packet loops and have a more fair and reasonable distribution of the packets over the network. In short, a packet would not be sent to a node where it had already been. Such mechanism makes sense when there are replications of the data packets in the network. But when it is supposed to exist only a copy of the packet in the network, forbidding it to go through a node where it has already been can introduce delivery problems. A few situations where that happens are the following:

- The packet was going through some hops in direction to the GW, but a node where it has already passed through is elected GW before reaching the previous GW. This means that this packet is now going back to a node where it has already been. The Loop Avoidance will not let that happen, leading to a very big delay on its delivery (wait until the GW is not a node where the packet has already been, or does not have nodes where it has been in the path to the GW), or not even ever delivering it.
- The packet passed through a node that was in the path from the sender to the GW, but due to mobility reasons, a node that had previously the packet is now in the path to the GW. The Loop Avoidance mechanism would, once again, not let the packet to be transmitted through this node.

Therefore, several modifications were made on this protocol, while maintaining the same strategy. Figure 4.3 presents the process from the senders perspective.

Figure 4.4 presents the process flow on the receiver side. After receiving the packet, an Acknowledgment packet will be sent to the sender confirming the reception of the data packet, and also confirming if it reached the final destination. The data packet's sender, after receiving the Acknowledgement, will delete the packet from the storage. The receiver will update the received packet hop list and store it, if it is not already in storage.

4.2.4 Neighbor's Energy Table Update

As the USVs are battery-operated they have a limited power supply, so it is important to have an efficient utilization of their energy. The GW of a cluster requires a higher expenditure of energy when compared to a regular node in the network, mainly because it will be receiving the data from all the nodes within the cluster, and will also forward that information to the BS. Also, being in the way to the GW requires more energy from this node, as it will be forwarding more packets than the average nodes.

The parameters used for the energy calculation to be used in equations (2.9) and (2.10) are presented in Table 4.1.

In order to keep a track of the energy of all the neighbors in the cluster, a table with the energy of each node is passed by through the beacons that are sent as *neighboring announcements*. This way, the information of the node's energy is spread through the cluster on a passive way, without a noticeable increase of the overhead on the network, neither requiring additional resources. It is

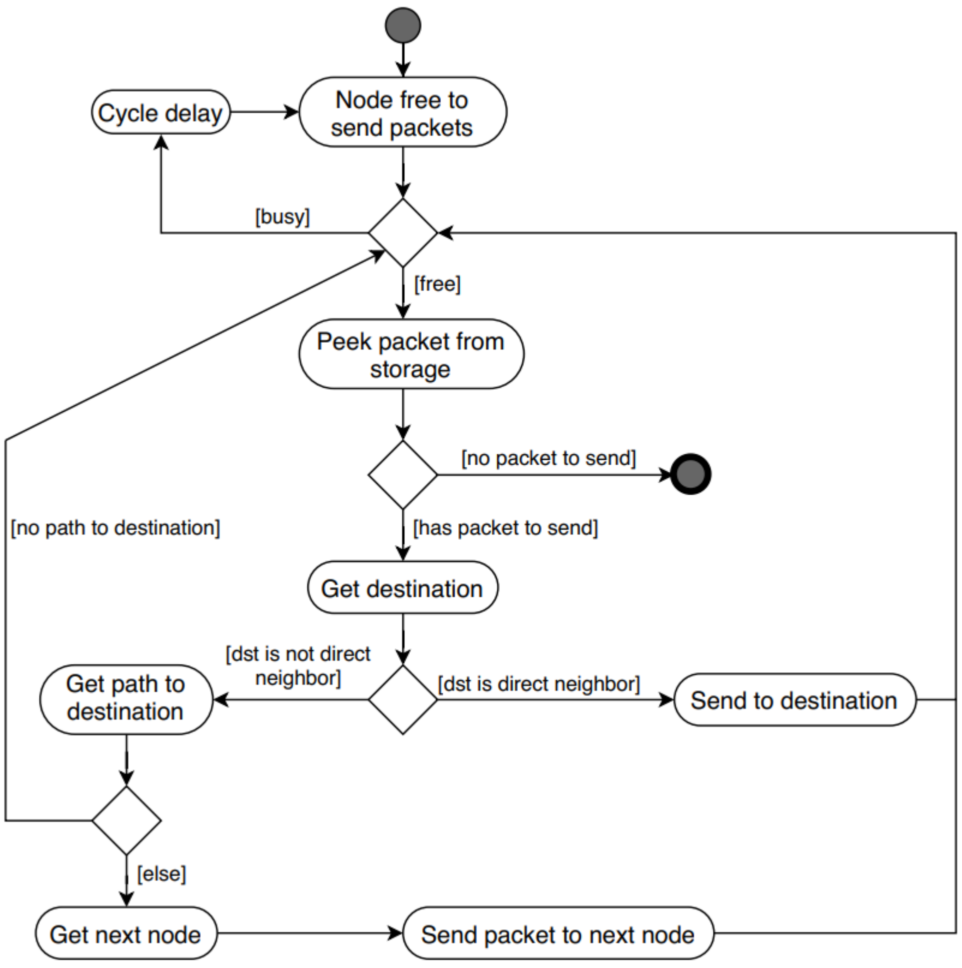
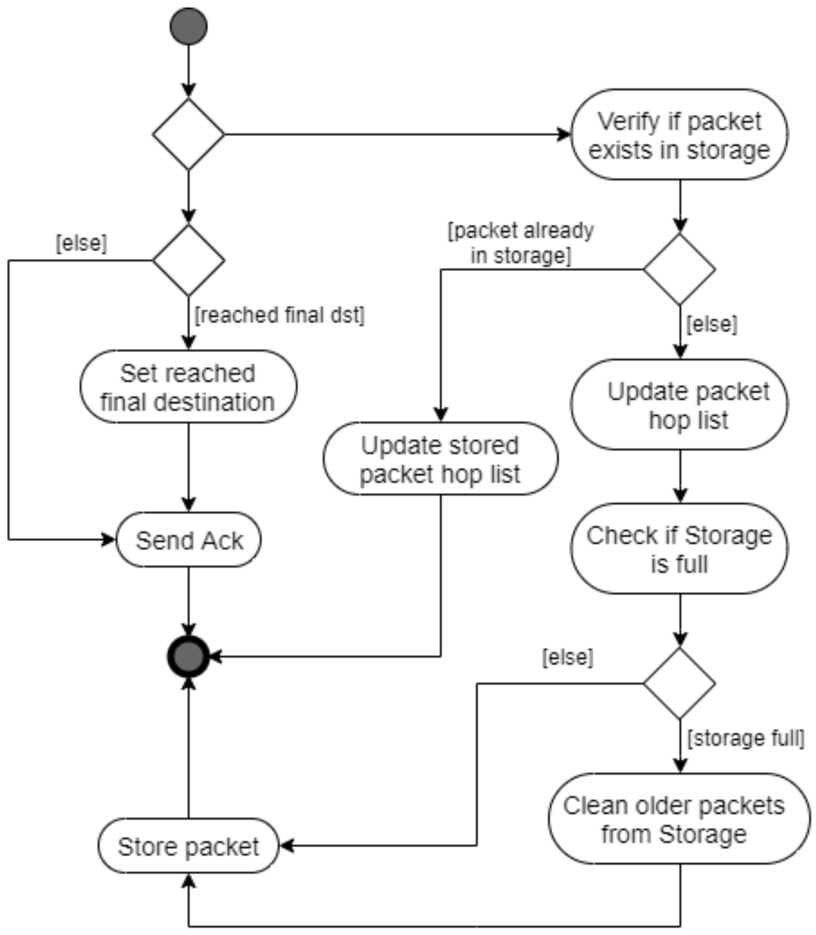


Figure 4.3: Data Sender Process Flow.

Parameter description	**Symbol**	**Values**
Energy dissipated in electronic circuit	E_{elec}	50 nJ/bit
Energy dissipated by transmission amplifier	ε_{amp}	100 $pJ/bit/m^2$
Path loss intra-cluster	λ	2
Path loss to BS	λ	2.5
Starting energy dedicated	E_0	100 J

Table 4.1: Parameters for wireless energy calculation [7].

spread and updated on a similar way as the network graph. The only difference is that the node's own energy is updated every time a packet is sent or received by the NIC. Such is done through an internal message that informs the Neighboring Layer and also indicates if the communication was inter-cluster or to the base station.



Verify if packet exists in storage
[else]
[packet already in storage]
[reached final dst]
[else]
Set reached final destination
Update packet hop list
Update stored packet hop list
Send Ack
Check if Storage is full
[else]
[storage full]
Clean older packets from Storage
Store packet

Figure 4.4: Data Receiver Process Flow.

4.3 Gateway Election

4.3.1 Clustering Process

The USVs are usually deployed as a team into an aquatic environment. During the monitoring task, the initial team (or the initial cluster) may be divided in smaller clusters. Clustering the network increases the lifetime and quality of the network [29]. A cluster is a group of nodes that can reach each other through one or more hops, and have elected a cluster head, the GW, to which they send the collected information (as represented in Figure 4.1). The GW gathers this information and sends it to the BS through a NIC that has a long range communication capability, or a range enough to reach the BS.

4.3.2 Gateway Rank Calculation

The deployed USVs should be able to, during their mission, change the cluster they belong, or to create a new cluster, while keeping a connection (direct or by some hops) to a GW. Many GW election methods are manly focused on the energy of each node. They also increase the overhead of the network, as they send additional packets. Our proposal is a passive election method, taking advantage of the neighborhood adjacency graph and the energy table already spread within the cluster for the rank calculation, allowing each node to have the knowledge of the network status in present time, and therefore, the nodes can elect the same GW every time a new GW is to be elected.

Based on the neighbors' graph weight and the neighbors' energy table that each node has stored, where i represents a node, the rank calculation for each node is made as follows:

$$Rank(i) = (1 - C_B(i)) \times LQ_{av} + C_B(i) \times Ener(i) \tag{4.3}$$

where the LQ_{av} is the average link quality between a node and the other nodes in the cluster which is normalized between 0 and 100. From the weight of each vertex v of the neighbors' graph we have the link quality estimation between two nodes, $LQ(v) = 100 - Weight(v)$. The N is the total number of nodes in the cluster and the LQ_{av} is calculated as follows:

$$LQ_{av}(i) = \frac{\sum_{n=0}^{N} LQ(n)}{N} \tag{4.4}$$

the $Ener(i)$ is obtained from the neighbors' energy table, being a percentage value with a minimum of 0 and a maximum of 100 with the value of energy on the node i. The $C_B(i)$ is the betweenness centrality presented by Ulrik Brandes [52], that quantifies the number of times a node acts as a hop along the shortest path between two other nodes. The betweenness centrality of a node i is given by the expression:

$$C_B(i) = \sum_{s \neq v \neq t \in V} \frac{\sigma_{st}(i)}{\sigma_{st}} \tag{4.5}$$

where σ_{st} is the total number of shortest paths from node s to node t and $\sigma_{st}(i)$ is the number of those paths that pass through i. Its output has a minimum value of 0 and a maximum of 1.

With this rank equation, we can have a result derived from the energy and link quality connection, where the weight of each factor in the equation is defined by the betweenness centrality. Hence, when a node has a higher betweeness centrality, the energy will weight more on the equation. That is because, due to this node's centrality, it will probably forward more messages between other nodes and a GW, when compared with a more isolated node, resulting on a higher energy expenditure. On the other hand, if a node as a lower centrality, it has a higher probability of having a poorer link quality connection within the network. So, here the link quality of the connection should have more impact on the decision, and on the ranking metric.

4.3.3 Gateway Election Process

The GW of a cluster might change over time, specially under a mobile environment. Therefore, a cluster must be able to elect and re-elect a GW.

For the election of a cluster's GW, two methods are presented that will propose a GW to be elected, and three control strategies that will decide if the proposed GW is, then, elected. One method, based on the energy level of each node, is similar to the most common GW election strategies, and it will be used as baseline for the second proposed method. The second proposed method is based on the ranking metrics previously presented and represented by equation (4.3), that uses not only energy, but also the link quality of connections between the node and its neighbors. It also takes into account the betweenness centrality of the node in the cluster.

Besides the two GW election methods, two control strategies are also introduced. These are responsible to control if or when a new GW is elected. The control strategies verify the method's results and decide if their proposed GW gets elected or not.

This GW election process is passive, as it is done based on the metrics passed along with the beacons (neighborhood graph and energy table), taking advantage of the fact that each node knows the cluster's network state. Therefore, it is possible to elect a new GW without the necessity of introducing extra messages (more overhead) in the network. In addition to electing individually the GW by each USV, the elected USV should not fail on electing itself as GW, otherwise, the elected GW by the cluster never behaves as GW.

4.3.3.1 Method 1: Energy-based

Algorithm 1 represents the rationale behind the method 1, an energy-based heuristic. It starts by going through all the nodes in the cluster, calculating for each node its betweenness centrality and energy (lines 1 to 3). Then, it compares their energy until it finds the node with the highest energy value (lines 7 and 8). If two nodes have the same energy value, the tiebreaker is done by choosing the node that has the biggest betweenness centrality (lines 9 and 10). If they are tied again, the node with the lowest ID is the chosen one (line 12).

Algorithm 1: Method 1: energy-based

```
1  for id = first node to id = last node of the cluster do
2      Cb(id) ← get_central_betweenness(id) ;            ▷ $C_B$ from eq (4.5)
3      Ener(id) = EnergyTable(id) ;
4  best_id = -1;
5  highest_energy = 0;
6  for id = first node to id=last node of the cluster do
7      if Ener(id) > highest_ energy then
8          best_id=id;
9      else if Ener(id) == Ener(best_id) and Cb(id) ≠ Cb(best_id) then
10         best_id = node id with the highest betweenness centrality;
11     else
12         best_id = node with the lowest id;
```

With this method, we expect the GW to vary frequently, keeping a uniform value of battery within the cluster, but having a higher delay, as it will be electing mostly with the same frequency all nodes overtime, regardless their poor quality of connection to the neighborhood.

4.3.3.2 Method 2: Rank-based

This method is based on a ranking metric, in which the energy and the link quality of the connection between the node and the remaining nodes in the same cluster are the only metrics for the classification, where the betweenness centrality decides which metric should have more weight upon the final decision.

As shown in Algorithm 2, the rationale behind the method 2 is to start by computing the betweeness centrality for each node in the cluster. Then, the average link quality of each node in the cluster to the others is also calculated, and, after that, the rank (Equation (4.3)) of each node. Next, it compares the rank of each node in the cluster until it finds the node with the best rank. If two nodes end up having the same rank value, the tiebreaker is done by selecting the node with the biggest betweenness centrality. If they are once again tied, it is chosen the node with the lowest ID.

With this method, we expect that the GW will vary less frequently than the previous method. A major difference of battery levels within the cluster is expected, specially from nodes with higher centrality. A lower delivery delay is expected, as it will be electing nodes with better link connection to the neighborhood.

Algorithm 2: Method 2: rank based (4.3)

```
1  for id = first node to id = last node of the cluster do
2      cb(id) ← get_central_betweenness(id) ;                        ▷ $C_B$ from eq (4.5)
3      av_lqe(id)= ← get_average_lqe(id) ;                           ▷ $LQ_{av}$ from eq (4.4)
4      ener(id) = EnergyTable(id) ;
5      rank(id) ← get_rank(id, cb(id), av_lqe(id), ener(id))         ▷ Rank from eq (4.3)
6  best_id = -1;
7  highest_rank = 0;
8  for id = first node to id=last node of the cluster do
9      if rank(id) > highest_rank then
10         best_id=id;
11     else if Rank(id) == Rank(best_id) and cb(id) $\neq$ cb(best_id) then
12         best_id = node id with the highest betweenness;
13     else
14         best_id = node with the lowest id;
```

4.3.3.3 Control Strategies

After the selection of a new GW, and in order to control the periodicity of a GW exchange, two control strategies were proposed. Additionally, a third control strategy was proposed to be executed when there is no elected GW in the node.

A periodic GW check up event is scheduled to occur every *gwCheckPeriod* seconds, defined previously. The schedule is made after the election of a new GW, or after a previous check up that has not scheduled any other kind of control check up event. Within this check up, from the previously presented methods, a GW is proposed to the control strategies, that, lastly, will decide if the proposed GW is elected.

The first control strategy is, actually, not having a real control on the frequency of the GW election. If a new GW is proposed by the chosen method, then it just checks its address and directly elects it without any additional control, scheduling at the end the periodic GW check up event.

The second control strategy is presented in Algorithm 3. Here, when a new GW is proposed to be elected by the method, it is rechecked a defined number of times whether this node must be proposed as GW in a row (*control_ count*) (lines 3 to 6). This is done by saving the new proposed GW and scheduling a recheck up event in *control_ time* seconds to verify if it is proposed again (lines 13 and 14). After suppressing the *control_ count* value (line 7), if the rank of the proposed GW is at least 10% better than the rank of the current GW (line 9), then the proposed new GW is the elected GW and the periodic GW check up event is set (lines 10 and 11). This control prevents an uncontrolled change of the GW overtime, as it has to be at least 10% better than the previous one. It also avoids situations where nodes elect different GWs, as this control strategy makes the node recalculate the proposed GW a few times before concluding that it is the best choice. This recheck up also gives time for the node to receive new information, that might be slightly delayed, about the neighborhood, decreasing the chance of wrong GW elections by the nodes.

Besides these two control strategies that we can decide to use prior to the deployment of the node, there is an additional control strategy that the GW election process goes through

Algorithm 3: Second Control Strategy

```
count_new_ElectGw = 0;
if exists a current GW and the proposed GW $\neq$ current GW then
    if new GW == GW proposed on the last check then
        count_new_ElectGw++;
    else
        count_new_ElectGw = 0;
    if count_new_ElectGw > control_count then
        count_new_ElectGw = 0;
        if rank of new GW > 0.9$\times$rank of current GW then
            elect new GW as current GW;
            schedule next periodic GW check up event;
    else
        save new GW as GW proposed by this check up;
        schedule a new GW recheck up event in control_time seconds;
else
    schedule next periodic GW check up event in gwCheckPeriod seconds;
```

when there is not a current GW elected. It is similar to the previously presented second control strategy, and it is represented in Algorithm 4. Not having a current elected GW can occur at the beginning, when the nodes are deployed without a predefined GW, or when the cluster suffers a change on its elements, that is, a node or more leaves or enters the cluster. Such action forces a deletion of the current GW on each node, so it triggers a new GW calculation and election, taking into account the new state of the cluster. This control strategy rechecks if the proposed GW is proposed more than *control_ countP* times in a row (line 7), scheduling this recheck up in periods of *control_ timeP* seconds (line 13). After overcoming these steps, it elects the proposed GW (line 8). These control parameters are defined in the configuration of the simulation, and also prevent the election of different GWs in the same cluster by its nodes.

Gateway Election Process Flow

Figure 4.5 shows the full process of the GW election, where the previously presented methods and control strategies are contained. First, it checks if the node is alone in the cluster or if it has neighbors. If it is alone, it rechecks this state 5 times in a row, separated by *gwCheckPeriod* seconds each, until it elects itself as a GW. However, if it has neighbors, it calculates the rank for each node accordingly to the method that was implemented. If the cluster suffered any change since the last GW check up, that is, if there are new nodes in the cluster or if any node has left, it deletes the current GW. This forces a new election without passing through the first control strategy (the control strategy we decided to use from the two control strategies previously presented). In this situation, it goes through the additional (third) control strategy, where it checks if the method proposes the same GW more than *control_ countP* times in a row, rechecking in periods of *control_ timeP* seconds. When so, it elects the proposed GW as current GW. However, if the cluster has not changed, it applies the chosen control strategy and, if so decided by the control strategy, it elects a new GW. After a new GW is elected, the periodic GW check up event is schedule in *gwCheckPeriod* seconds.

The *control_ time*, *control_ timeP*, *control_ count*, *control_ countP*, and *gwCheckPeriod* are parameters of the control strategies and are defined at the beginning of the simulation. They can

Algorithm 4: Third (fixed) Control Strategy

```
count_new_Gw_check = 0;
if *there is no current GW* then
    if *new GW = GW proposed on the last check* then
        count_new_Gw_check++;
    else
        count_new_Gw_check=0;
    if *count_new_ElectGw > control_countP* then
        count_new_Gw_check = 0;
        elect new GW as current GW;
        schedule next periodic GW check up event;
    else
        save new GW as GW proposed on this check;
        schedule new GW recheck up event in *control_timeP* seconds;
else
    schedule next periodic GW check up event;
```

be changed for a better or worse performance of the proposed methods and control strategies. In section 5 it is presented a scenario where these parameters are changed and their results are analysed.

4.4 Chapter Considerations

In this chapter it was presented the proposed work and the fundamental implementations for its development. We start with an overview of the aquatic architecture that requires the implementations made in this work. It was described the implementation of forwarding mechanisms that allows the gathering of the data and deliver to the BS, based on LQE metrics. It was also presented the energy table that tracks and shares the energy state of each node within the cluster. Finally, it was presented the proposed and implemented methods for the GW election within a cluster based on different factors, and the control strategies that will control the GW election outcome based on predefined parameters.

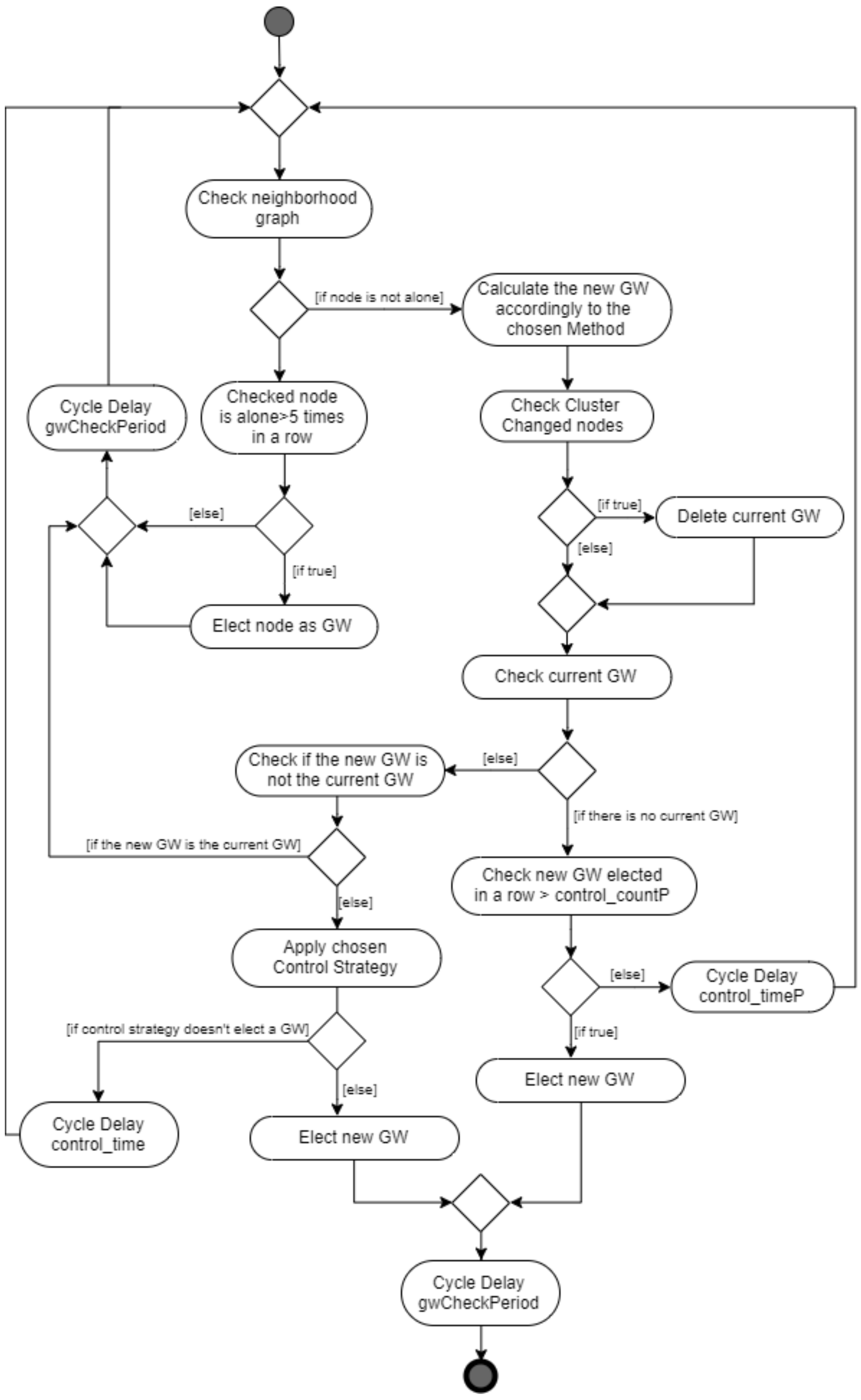

Check neighborhood graph
[if node is not alone]
Calculate the new GW accordingly to the chosen Method
Cycle Delay gwCheckPeriod
Checked node is alone>5 times in a row
Check Cluster Changed nodes
[else]
[if true]
Delete current GW
[else]
[if true]
Elect node as GW
Check current GW
Check if the new GW is not the current GW
[else]
[if there is no current GW]
[if the new GW is the current GW]
[else]
Check new GW elected in a row > control_countP
Apply chosen Control Strategy
[else]
Cycle Delay control_timeP
[if control strategy doesn't elect a GW]
[if true]
Elect new GW
[else]
Cycle Delay control_time
Elect new GW
Cycle Delay gwCheckPeriod

Figure 4.5: Gateway election flow.

Chapter 5

Tests and Results

Once designed and implemented, the simulator and the GW election mechanisms and control strategies need to be evaluated. This chapter presents the obtained results and the consequent performance analysis. Section 5.1 describes the equipment and software used in this work. Section 5.2 evaluates the overall behaviour of the proposed simulator. Section 5.3 describes the scenarios used on the evaluation of the proposed GW election methods and strategies on a DTN, and analyses the results. Section 5.4 presents the chapter summary and final considerations.

5.1 Equipment and Software

The following section describes the equipment and software used to develop and test the simulator, and also to evaluate the implemented work. Running simulations can be very demanding, so the time and capability to take determined tests depend on the hardware and software used. When running the simulator, for a better performance, without unknown problems, it is recommended to use the versions of the software which the simulator was developed upon.

All of the presented work was performed in a laptop with the specifications illustrated in Table 5.1.

Processor	Intel® Core™CPU i5-8300H @ 2.30 GHz x8
Memory RAM	8 GB
Graphics	GeForce GTX 1050
Operating System	64-bit Ubuntu 16.04 LTS

Table 5.1: Specifications of the machine used to develop and run the simulator.

The OMNeT++ along with the INET framework versions are v5.4.1 and v4.0.0, respectively. It is recommended to install these versions when running the developed simulator, as it might be incompatible with other versions.

5.2 Simulator Validation

This section evaluates the developed simulator.

5.2.1 The Validation Process

It is part of the process of developing a simulation system to have a stage of validation. This stage compares the performance model with the output of the real system. However, when researching in DTN networks, real networks might not be available, hence the need of a simulator. So, the validation must follow other strategies to compare the output of the simulations [53].

A first approach would be to consider the results obtained by mathematical analysis based on the same performance model. Unfortunately, this process can get too complex and difficult, specially in DTN networks, as they present some degree of random processes due to their opportunistic environment.

Another approach would be to consider previously published performance results, obtained either by analysis or simulation. This is also not easy, because many authors do not provide their code, neither describe all the metrics, parameters and values used.

A previous work in [3] is the closest one to the base architecture developed in the proposed simulator, since it has a similar DTN architecture and a similar passive LQE based forwarding strategy. The results from [3] referring to the PAmuLQE-NACK and Epidemic protocols tested in a simulated environment will be used for comparison. A different method for the validation is to compare the results obtained by the implemented LQE-based protocol with the Epidemic routing protocol, as it is a quite simple and known protocol, with logical expected results for simple scenarios, and that was also used in the work developed by [3].

The results are expected to present the same outcome as [3], but differ in terms of precision. Such happens for the following reasons: the hardware is different, both on the simulator and of the emulated USVs; the architectures of the simulators are conceptually similar but have some differences; the forwarding protocol is in both cases LQE-Based, but has some differences on its implementation; the presence of unknown variables, such as, sometimes, the total time of simulation and introduced delays within the architecture flow or routing decisions. These factors are noticed on the results mainly in terms of "during time results", i.e. , the simulation might vary in response over time. But at the end of each test, the overall results and the overall behaviour should be similar.

5.2.2 Scenarios Tested

In order to be able to compare the results, two scenarios presented by [3] were used, one static and another with mobility. The static scenario is illustrated in Figure 5.1. This scenario contains 3 USVs (nodes), all static. The USVs 1 and 2 will be generating 100 data packets each, that are to be transmitted to the USV 3, which is the GW USV. Also, USVs 1 and 3 are not directly connected, which means that the USV 2 acts as a connection point between USV 1 and 3. USV 2 is positioned 20 meters away from USVs 1 and 3.

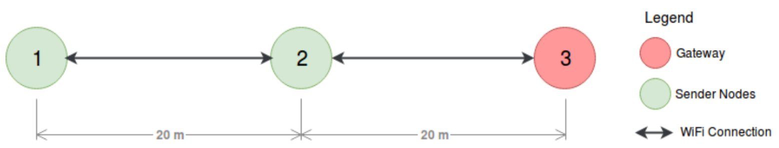


Figure 5.1: Aquatic static scenario [3].

The scenario with mobility is illustrated in Figure 5.2. In this scenario all USVs are mobile. The USV 3 is set as GW, while USVs 1, 2 and 4 generate data packets every second during 90 seconds of simulation, that are to be transmitted to the GW.

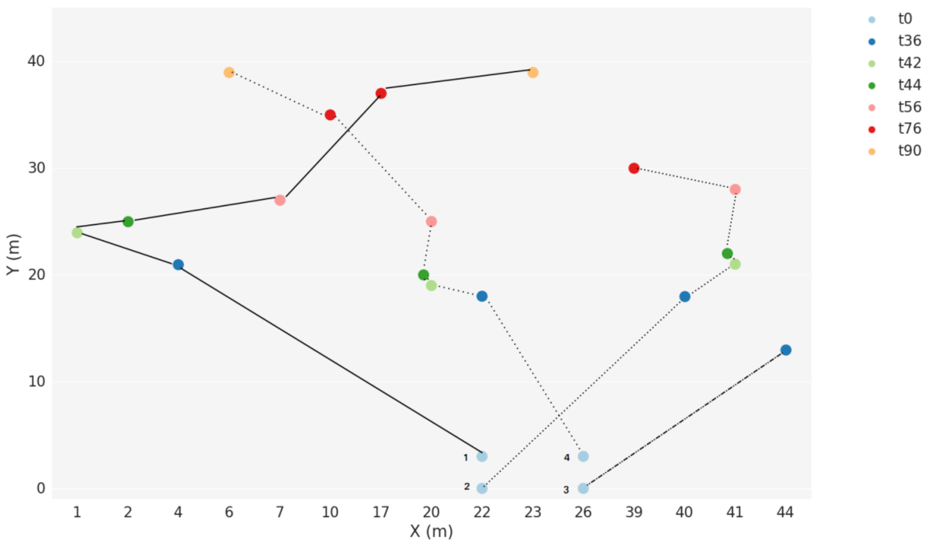


Figure 5.2: Position progress of each USV in aquatic scenario with mobility [3].

5.2.3 Comparison and Evaluation of Results from Simulation

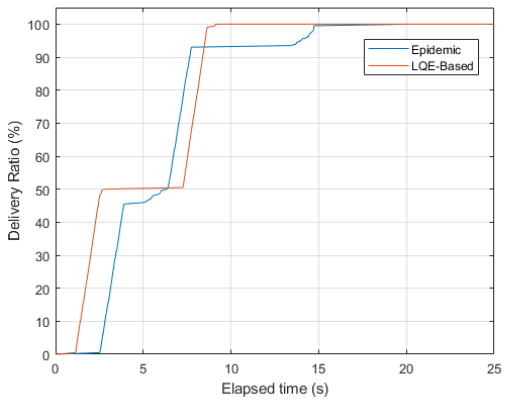
Static scenario

Figure 5.3 presents the overall network delivery ratio obtained in this scenario. As expected, the results from the proposed simulator differ from [3] in terms of the precision of their results, mainly because of the hardware and software differences that introduce architectural delays unknown by us. Altogether, they present the same overall success. Both Epidemic and LQE-Based protocols behave similarly, just as it happened in [3], achieving the 100% of delivery rate, though in our simulator it is achieved earlier. For this reason, the mean delay, illustrated in Figure 5.4, is closely the same but differ from [3] because of the aforementioned aspects. The bar plots represent the average amount of time that a data packet takes from its generation to the time it reaches a GW. The redundant overhead is presented in Figure 5.5. The bar plots represent the amount of redundant data packets that each strategy introduces in the network. As expected, the Epidemic introduces a much higher redundant overhead than the LQE-based strategy. That is because the Epidemic strategy broadcasts the packets, which leads to a high replication rate.

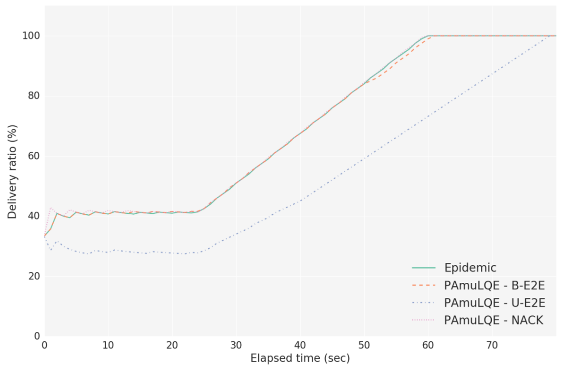
Scenario with mobility

In the scenario with mobility, as illustrated in Figure 5.2, all USVs change their position over time, leading to a change of connections between the USVs, and consequently, a change of paths to the GW. USV 3 acts as a GW, while USVs 1, 2 and 4 generate packets to be transmitted to the USV.

Once again, the results are within what was expected. In Figure 5.6 we can observe that the delivery ratio reaches the 100% after some seconds and stays near this value, having some fluctuation due to the delays caused by the USV's mobility and the consequent change of the



(a) AquaticDTNS.



(b) Network simulator in [3].

Figure 5.3: Delivery ratio in static scenario.

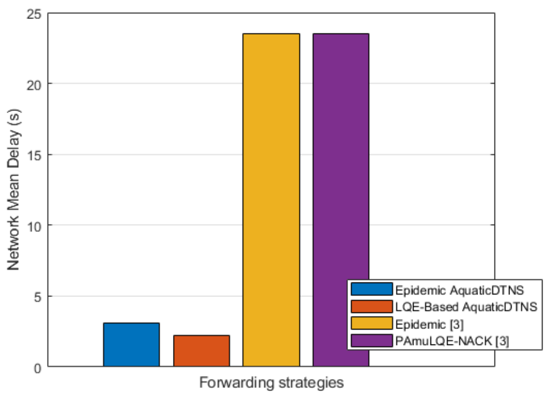


Figure 5.4: Network mean delay in static scenario.

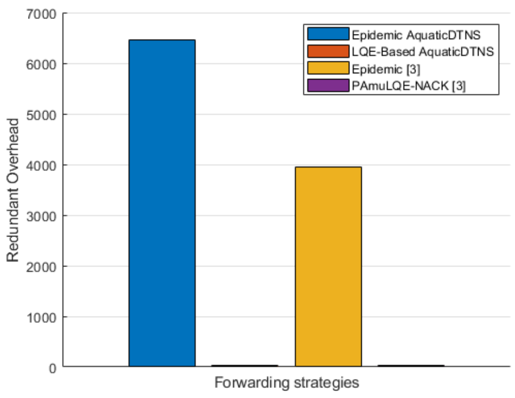
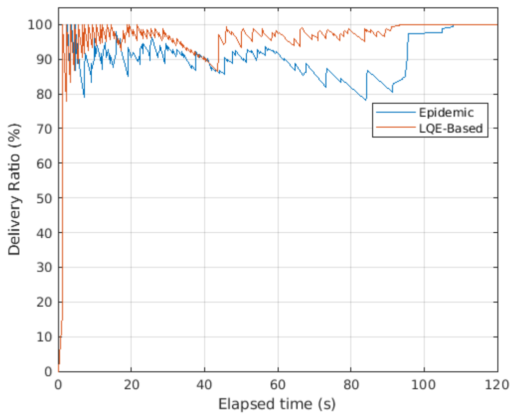


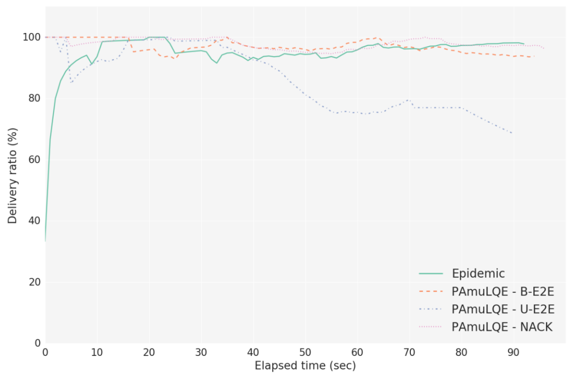
Figure 5.5: Network overhead in static scenario.

paths to the GW. Figure 5.7 presents the network mean delay, where the bar plots represent the average amount of time that a data packet takes from its generation to the time it reaches a GW. As expected from [3], the Epidemic strategy has a higher mean delay than the LQE-Based strategy. Figure 5.8 presents the network redundant overhead, where the bar plots represent the amount of redundant data packets that each strategy introduces in the network. The redundant overhead introduced in the network by the Epidemic strategy is also much higher than the introduced by the LQE-Based strategy.

Overall, from the results obtained by the presented simulations on different scenarios with different routing protocols, the proposed simulator behaves accordingly to what is expected. It reached the same success level, and, thus, presents itself reliable for testing different network mechanisms and protocols in DTN networks on aquatic surface environments.



(a) AquaticDTNS.



(b) Network simulator in [3].

Figure 5.6: Delivery ratio in scenario with mobility.

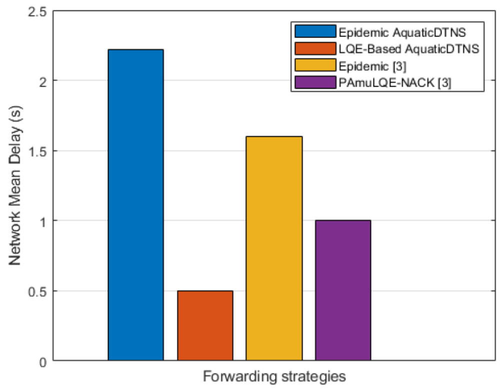


Figure 5.7: Network mean delay in scenario with mobility.

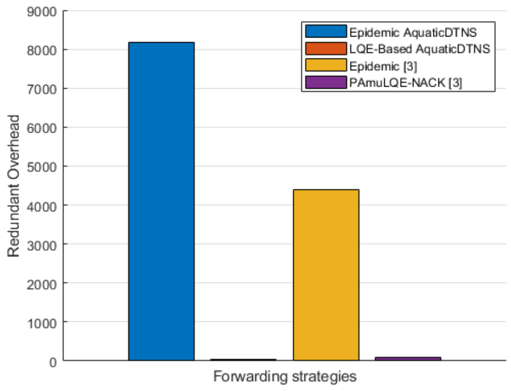


Figure 5.8: Network overhead in scenario with mobility.

5.3 Network Evaluation

In this section the developed simulator, that was validated previously, is used to evaluate the proposed GW election methods and control strategies through several tests on different aquatic monitoring scenarios.

5.3.1 Simulator Configuration Parameters

Table 5.2 presents the configuration parameters that were kept constant for all the simulations used to evaluate the proposed network mechanisms. Other parameters that have not been mentioned so far are either not used or will be mentioned before each simulation analysis, as they are different depending on the experiment.

Application module	
dataGenerationInterval	1 *s*
dataSizeInBytes	10 *bytes*
ThatGen	false (all nodes generate data)
startMultipleMsg	false
hopsListSize	50
Routing module	
maximumCacheSize	500 *kB*
useTTL	false
waitBFsend	0.1 *s*
gwCheckPeriod	5 *s*
Neighboring module	
N_ nodes	15
sendBeaconInterval	1 *s*
maximumRandomBackOffDuration	1 *s*
WifiFirst	true
max_ absent_ time	2 *s*
Wireless module	
wirelessRange	40 *m*
minSSI	-90 *dBm*
neighborScanInterval	50 *ms*
limitQueueTime	400 *ms*
bandwidthBitRate	100 *Kbps*
wirelessHeaderSize	32 *byte* (WiFi header [44])
Mobility module	
mobilityType	*BonnMotionMobility*
is3D	false

Table 5.2: Parameters of the configuration of the simulator.

5.3.2 Scenarios Description

Different scenarios were used to compare the performance of the different GW election methods and control strategies. Two scenarios (A and B) are static and with only one cluster, while scenario C contains mobility and forms two clusters. Scenario D represents, on simulation, a real and dynamic aquatic monitoring task, where different clusters are formed during run time with the USVs changing places between them.

Scenario A is illustrated in Figure 5.9. This scenario contains 5 USVs (nodes) with no predefined GW. All USVs are fixed: USVs 0, 1, 3 and 4 are positioned on the vertices of a square 20 meters away from each other, and USV 2 is precisely in the middle. They all have connection among each other and generate data packets to be transmitted to the GW. The BS is located 500 meters into land from where the USVs were deployed.

The topology of scenario B is illustrated in Figure 5.10. In this scenario the USVs 0, 1, 3 and 4 are now distanced 44 meters from each other and they have no direct connectivity. USV 2 is in the middle, also serving as a bridge of connection between the other USVs. All the USVs generate data packets to be transmitted to the GW. The BS is located 500 meters into land from where the USVs were deployed.

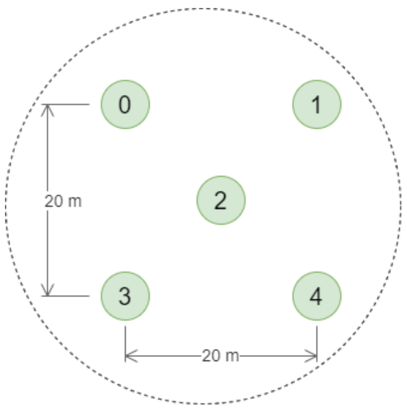


Figure 5.9: Aquatic scenario A.

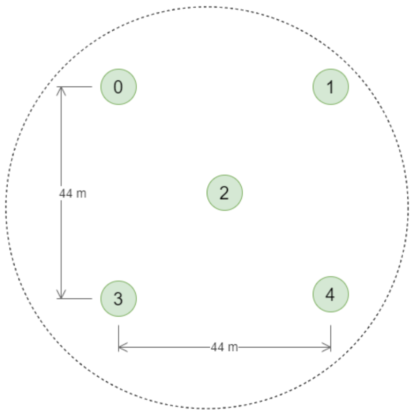
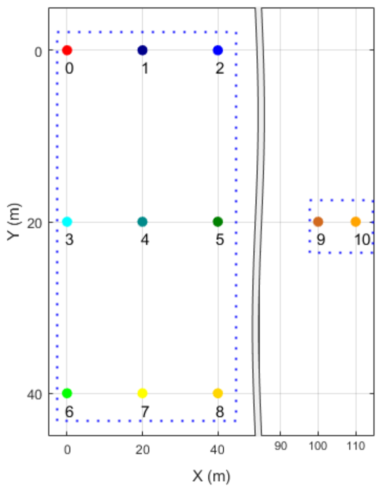
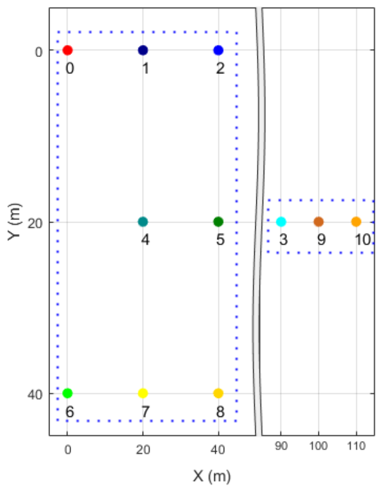


Figure 5.10: Aquatic scenario B.

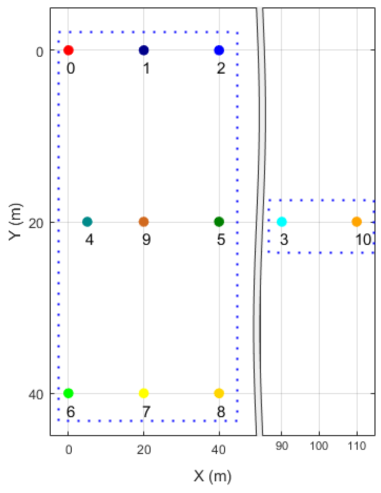
The topology of scenario C is illustrated in Figure 5.11. In this scenario there are 11 USVs divided in two clusters represented by the blue dotted squares. Initially, there are 9 USVs in one cluster and 2 in another one (Figure 5.11a). In the first cluster the 9 USVs are separated from each other 20 meters, and in the second cluster the USV 9 is 10 meters away from USV 10. USVs 3, 4 and 9 have mobility during this scenario. After 20 seconds of simulation time, the USV 3 moves in direction to the second cluster and reaches it at 40 seconds, as illustrated in Figure 5.11b. Then, at 70 seconds of simulation the USV 4 moves to the left within the cluster, and the USV 9 gets out of its cluster and joins the first cluster. Both USVs stop their movement at 80 seconds of simulation run time, as illustrated in Figure 5.11c. The BS is located 500 m into land from where the USVs were deployed.



(a) $T_0 = 0\ s$.



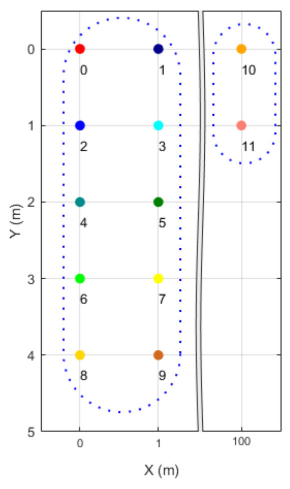
(b) $T_1 = 40\ s$.



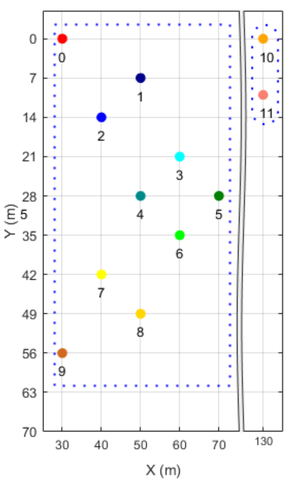
(c) $T_2 = 80\ s$.

Figure 5.11: Aquatic scenario C.

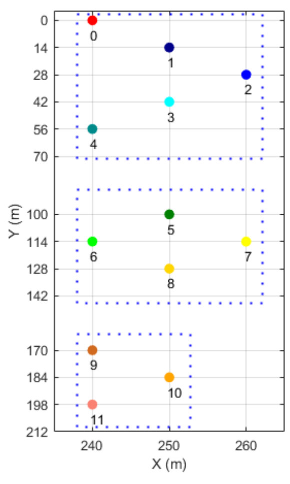
Scenario D mimics a real monitoring task, where the USVs change their position and clustering over time and. Its topology is represented in Figure 5.12 through several snapshots of their formation changes during time. The dotted line surrounds the USVs of each cluster. This scenario is made of 12 USVs that are initially grouped in two clusters, one of 10 USVs that are deployed together and another of 2 USVs that are already deployed 100 meters away, as shown in Figure 5.12a. All USVs move with an average speed of 1 m/s. After deployment, at 30 seconds of run-time of simulation, they are as represented in Figure 5.12b, and after two minutes of straight line movement, they change to the formation represented in Figure 5.12c. The same way, after 420 seconds of simulation, their positions are as shown in Figure 5.12d. At this point they stop moving away from their initial deployment zone and start coming back, changing to the formation represented in Figure 5.12e. The monitoring task continues with the same flow, passing through other two formations, first at 600 seconds and then at 780 seconds, as represented in Figures 5.12f and 5.12g. The latter formation is kept until the USVs stop their movement at 900 seconds, with the positions represented in Figure 5.12h. The simulation ends at 1000 seconds.



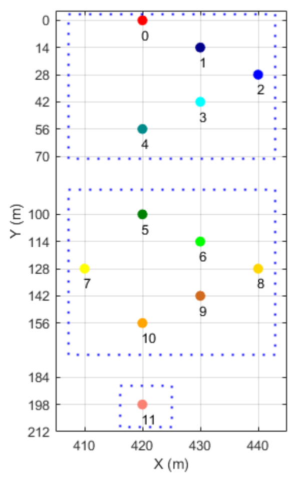
(a) $T_0 = 0\ s$.



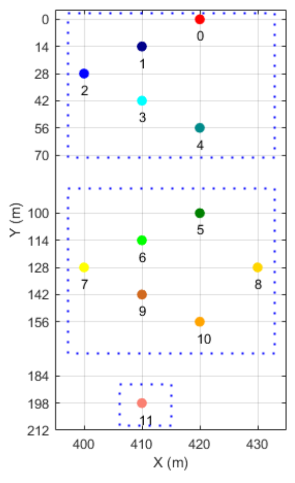
(b) $T_1 = 30\ s$



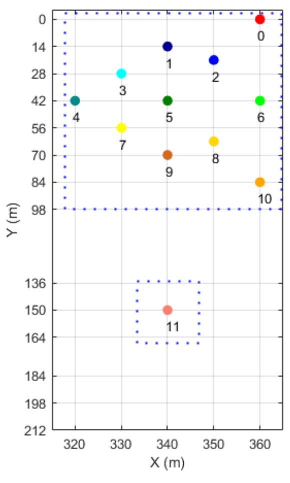
(c) $T_2 = 240\ s$



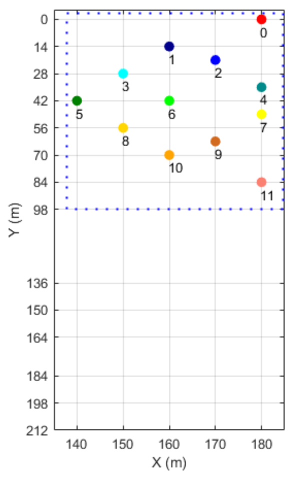
(d) $T_3 = 420\ s$



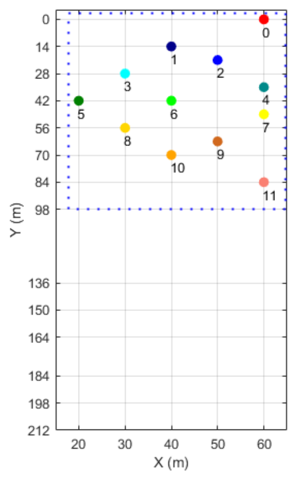
(e) $T_4 = 420\ s$



(f) $T_5 = 600\ s$



(g) $T_6 = 780\ s$



(h) $T_7 = 900\ s$

Figure 5.12: Aquatic scenario D.

5.3.3 Evaluation of the Different Methods and Strategies

The previously presented scenarios A, B and C are used to evaluate the two implemented GW election methods and control strategies, where the first method is referred to as $M1$ and the second $M2$, and the control strategies are referred, the first as $C1$, and the second as $C2$. For these tests, the configuration parameters presented in Table 5.3 complete the already described in Table 5.2, according to the respective scenario. During this section, the parameters of the control strategies will not change, having values with whom these tests would have a fair behaviour in order to analyse and compare the evaluated methods on each control strategy. The *control_ count* and *control_ countP* are the number of times the control strategies verify the proposed GW, distanced by periods of, respectively, *control_ time* and *control_ timeP* seconds, before electing the proposed GW. Their values are: $control\_count = 2$; $control\_time = 2$; $control\_countP = 5$; $control\_timeP = 1$.

General	
sim-time-limit	200 s
numNodes	5 (A & B); 11(C)
Application module	
endGeneratingMsg	150 s
Neighboring module	
EnergyStart	100 J
BS_position	(500,25)

Table 5.3: Extra parameters of the configuration of the simulator for the scenarios A, B and C.

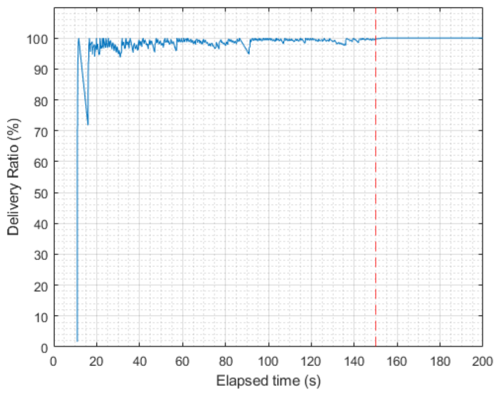
For the evaluation of the GW election methods and control strategies, the following results are presented:

1. the delivery ratio, that represents the ratio between the number of packets sent by the USVs and the ones that reached the GW;
2. the GWs that recognized their election, represented along the simulation time;
3. the number of events processed by the simulator;
4. the redundant overhead, that is the amount of data packets that are duplicated and reach the GW more than one time, that can occur due to loss of acknowledgment message reception, which makes the USV resend the data packet;
5. the mean delay, which presents the amount of time it takes for a message sent from the source to be received by the GW;
6. the total data hops, which is the sum of the hops that all the data packets received at the GWs went through from their sources;
7. the battery difference, which represents, from all the clusters, the biggest gap between the USV with the remaining highest energy and the USV with the lowest energy at the end of the simulation within the cluster.

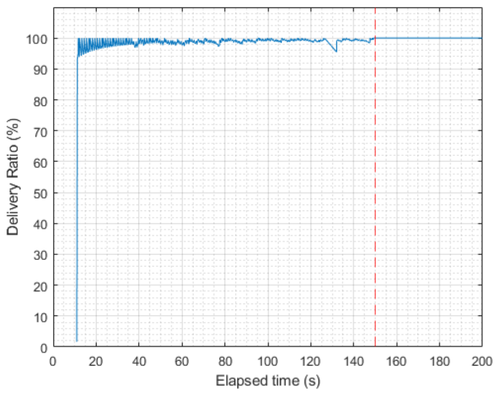
Scenario A

This scenario aims to evaluate the developed work on a simple aquatic monitoring environment. When using the first method, we can observe in Figure 5.13a that the delivery ratio has a higher variation at the beginning than in method 2 that is represented in Figure 5.13b. This is because, with this control strategy, the elected GWs change within short intervals. At the beginning, which is a period that normally requires a superior time to establish a stable delivery ratio since the network is being formed, the GW quickly changes (Figure 5.14a). With the second method, the first elected GW stays a bit longer (Figure 5.14b) as its election takes into account other parameters besides the energy, such as the betweeness centrality. The first elected GW is, as expected, the USV 2, the one with highest centrality. But after its rank decreases, the election starts varying just like with the first method due to the control strategy. Roughly after 150 seconds, when the USVs stop generating data, represented by the red dotted line, the elected GW remains unchanged. This is because its energy does not change as it has already delivered most of the generated data and are only exchanging beacons, which consumes less energy.

Table 5.4 presents the remaining evaluation metric results. In method 2, the delivery ratio is better right at the start, with a lower mean delay value. The battery difference at the end of the simulation is higher on this second method. This is because, as the first method only takes into account the USV's battery, it elects a new GW when the current GW drops its energy. Method 2 also takes into account other parameters for the GW election, which means that the energy takes a lower impact on the GW election, so at the end the USVs present higher energy differences than in the first method.

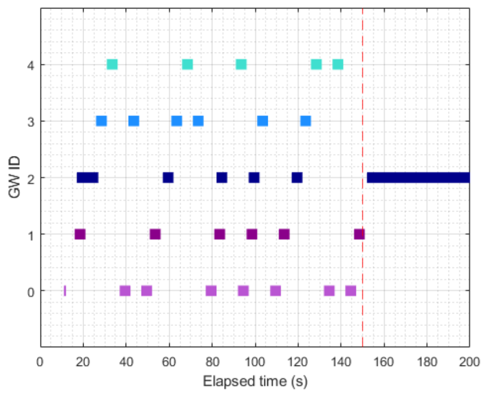


(a) M1C1.

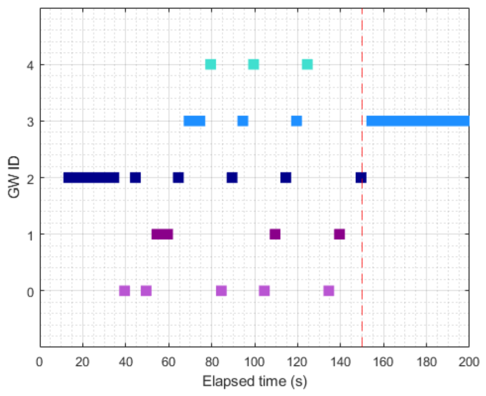


(b) M2C1.

Figure 5.13: Delivery ratio in scenario A with control strategy 1.



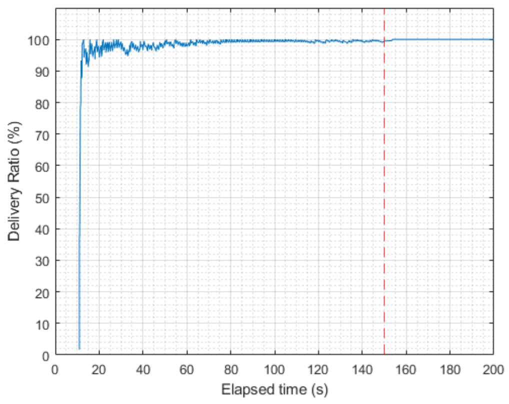
(a) M1C1.



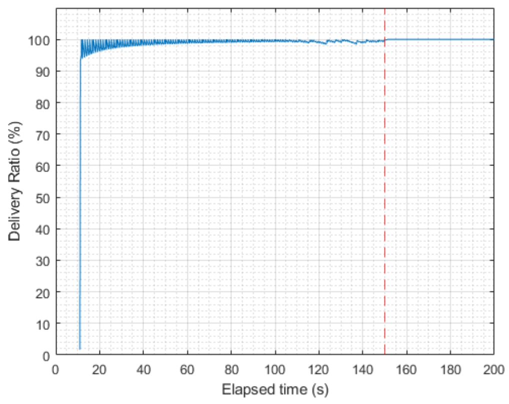
(b) M2C1.

Figure 5.14: Gateways elected in scenario A with control strategy 1.

By using the second control strategy for the GW election, as illustrated in Figure 5.16, the fluctuation of the GWs elected is much lower and controlled than the previous control strategy, resulting on a lower number of GW elections. From Figure 5.15 we can observe that, once again, the second method presents a slightly better delivery ratio and elects less GWs overtime. This is because it takes more time for the second control to elect a GW with a better rank calculated by the second method than the current GW rank, as it has to be 10% better for it to change GW.

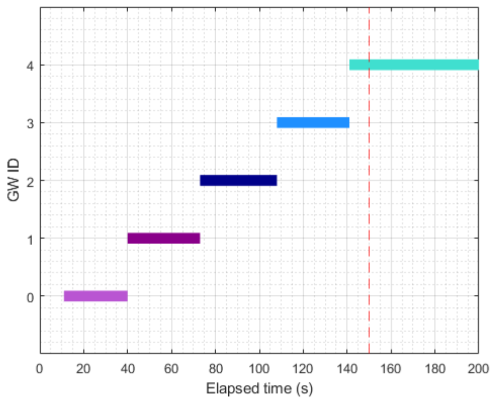


(a) Method 2.

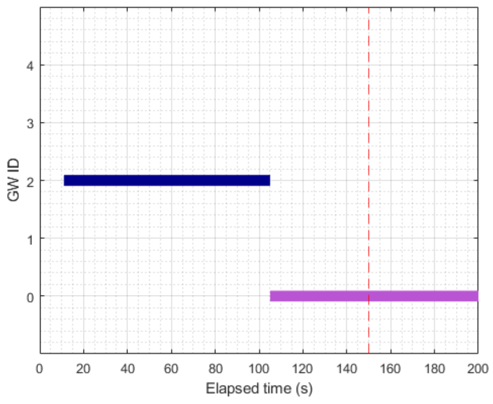


(b) Method 2.

Figure 5.15: Delivery ratio in scenario A with control strategy 2.



(a) Method 1.



(b) Method 2.

Figure 5.16: Gateways elected in scenario A with control strategy 2.

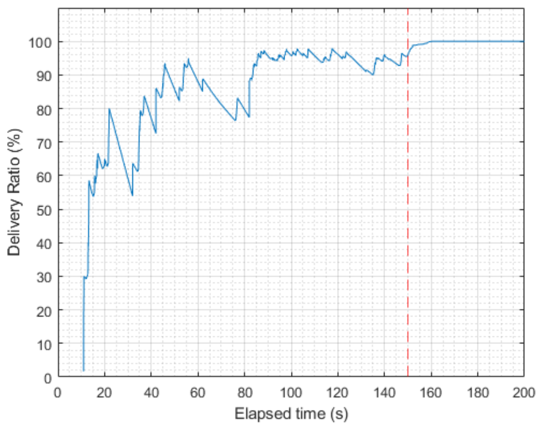
Scenario B

This scenario occupies now a larger area than the previous one. Only the USV 2 has direct connection to every USVs, so a higher delay and data hops are expected. The overall behaviour is similar to the previously presented scenario. In Figure 5.17 we can observe a better delivery ratio with method 2 than with method 1. As data messages are now not always delivered directly from the source to the GW, being retransmitted, the delivery ratio has a higher fluctuation throughout the time. Figure 5.18 shows that a high fluctuation of the elected GW occurs once again. Since in this scenario the USVs do not have direct connection among each other, except

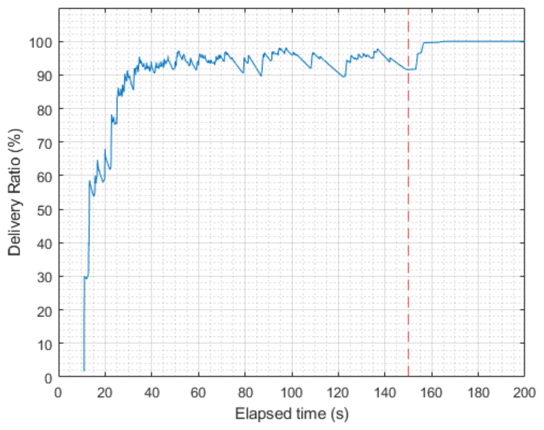
Parameter	M1C1	M2C1	M1C2	M2C2
Simulation events	364073	360380	360883	359421
Redundant overhead	25 pkts	11 pkts	0 pkts	0 pkts
Mean delay	2.4611 s	1.4847 s	0.3020 s	0.1825 s
Total data hops	815	645	602	600
GW elections	31	21	5	2
Battery difference	2%	15%	9%	34%

Table 5.4: Results for scenario A.

to USV 2, the first method also elects first the USV 2 as GW because at the beginning they all have the same level of energy, but now the USV wins the tiebreaker with its higher betweeness centrality, that was also equal among all USVs on the previous situation, where the tiebreaker used was the lowest USV *id*. From Table 5.5 we can observe, again, a better behaviour with the second method except for the battery difference that, as expected, is higher that when using the first method for the same reasons presented in scenario A.



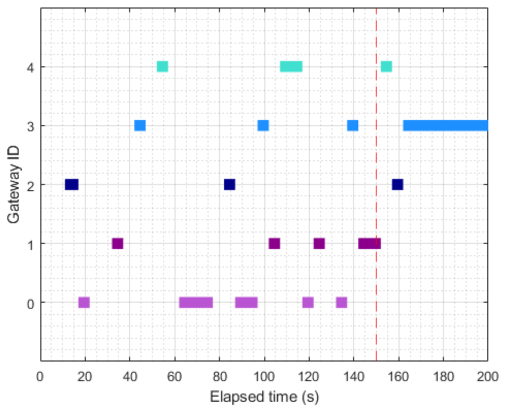
(a) M1C1.



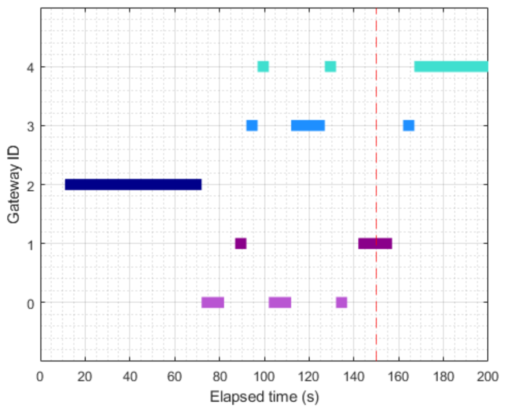
(b) M2C1.

Figure 5.17: Delivery ratio in scenario B with control strategy 1.

With the second control strategy, the overall behavior gets better as less GW elections occur on a short period of time. As illustrated in Figure 5.19 the delivery ratio gets better. Figure 5.20 shows that the election of GWs is more controlled, specially with the second method, providing more stability to the network. From Table 5.5 we can observe that the mean delay is much lower for both methods when using the second control strategy, a consequence of the lower number of total GW elections that allows a better delivery ratio of packets. The redundant overhead is also lower, which was expected as the network presents a better behaviour with the second control strategy. The first method has a lower battery difference than method 1, although it increases on both methods for this control strategy.

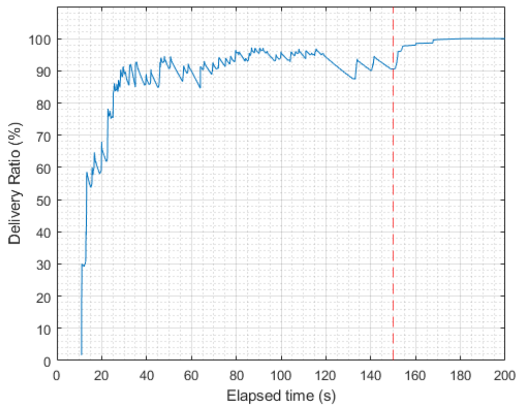


(a) M1C1.

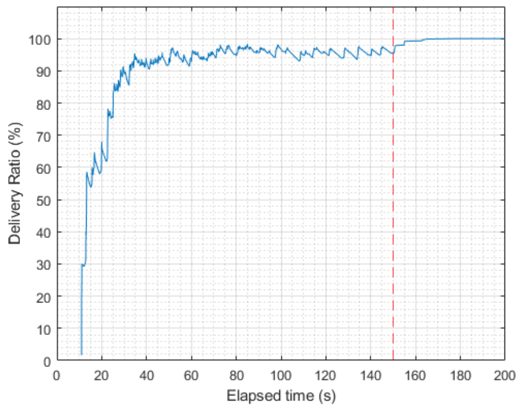


(b) M2C1.

Figure 5.18: Gateways elected in scenario B with control strategy 1.

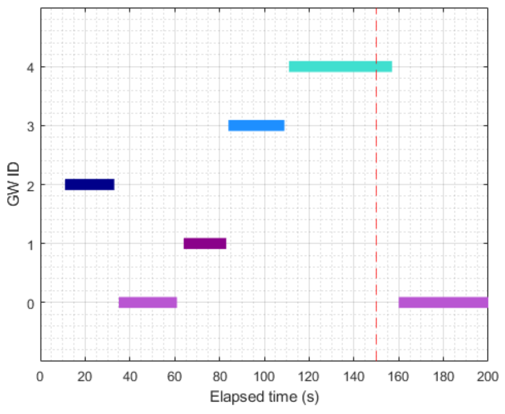


(a) Method 2.

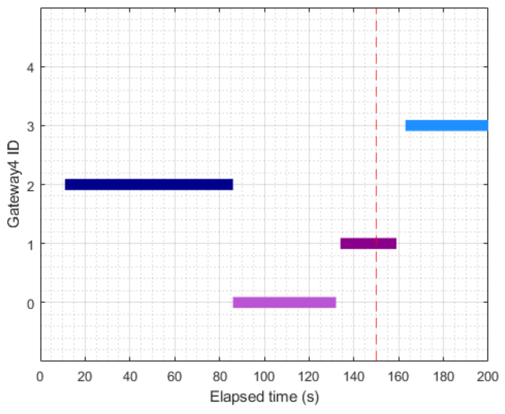


(b) Method 2.

Figure 5.19: Delivery ratio in scenario B with control strategy 2.



(a) Method 1.



(b) Method 2.

Figure 5.20: Gateways elected in scenario B with control strategy 2.

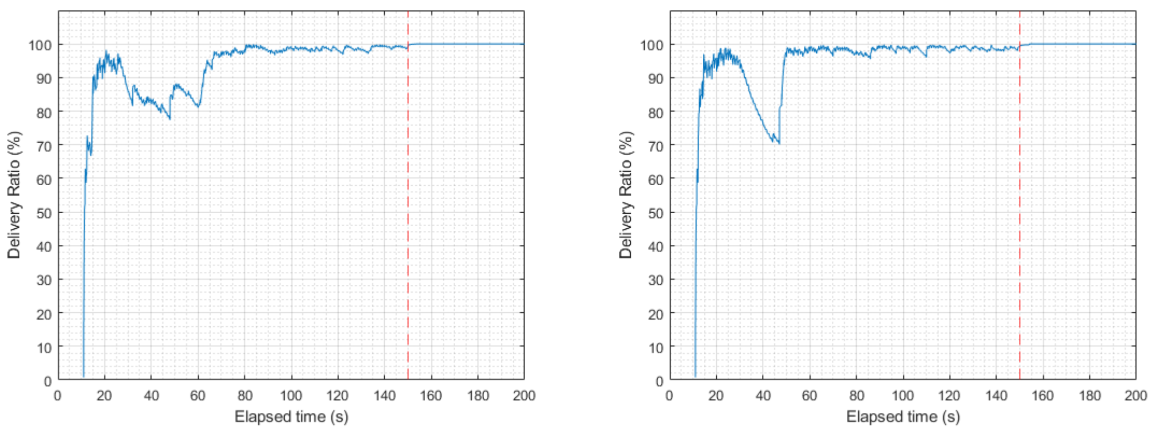
Parameter	**M1C1**	**M2C1**	**M1C2**	**M2C2**
Simulation events	275122	262219	269596	264182
Redundant overhead	299 pkts	144 pkts	101 pkts	45 pkts
Mean delay	12.7114 s	7.8166 s	4.7192 s	2.5038 s
Total data hops	1737	1069	1127	886
GW elections	19	12	6	4
Battery difference	7%	30%	13%	45%

Table 5.5: Results for scenario B.

Scenario C

This scenario aims to evaluate the developed work on an aquatic monitoring environment where two USVs switch between two clusters and with a much higher number of elements in the network.

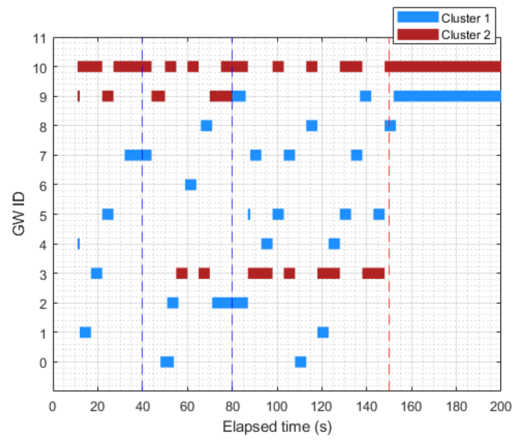
The first control strategy implemented for both methods results on a high number of GW election, as shown in Table 5.6 and illustrated in Figure 5.22. This results on an unstable delivery ratio on the periods that the USVs are in movement (around the 40 seconds), as illustrated in Figure 5.21. Also, from Figure 5.22, we can notice that there are GWs elected simultaneously within the same cluster or even the lack of a GW for a few time due to this chaotic change of GW. Such ends on a high number of total data hops and redundant overhead as the data messages are sent on the wrong direction more times.



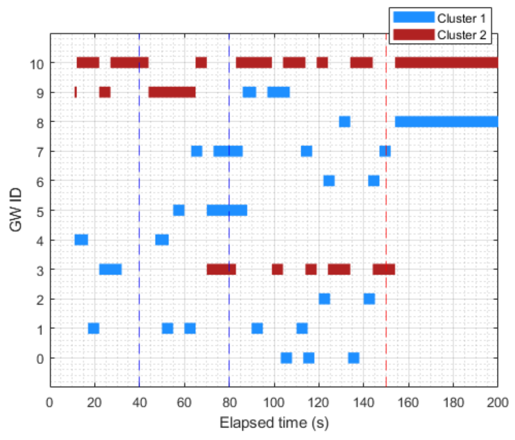
(a) M1C1. (b) M2C1.

Figure 5.21: Delivery ratio in scenario C with control strategy 1.

By using the second control strategy for the GW election, the overall behaviour gets much better. From Figures 5.23 and 5.24, it can be observed that the first method still presents a slight worst delivery ratio and a higher variation of the elected GW during time, which was expected as it presented a similar behaviour on the previous scenarios. From the results in Table 5.6 we can observe that, even though both methods present a similar mean delay, the first method has a higher number of total data hops. This was expected as the first method does not take into account the USV with the highest betweeness centrality and quality of connection, ending up electing USVs that are further away than the majority of the USVs.

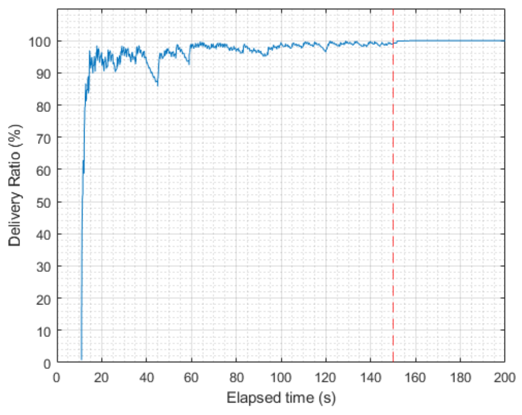


(a) M1C1.

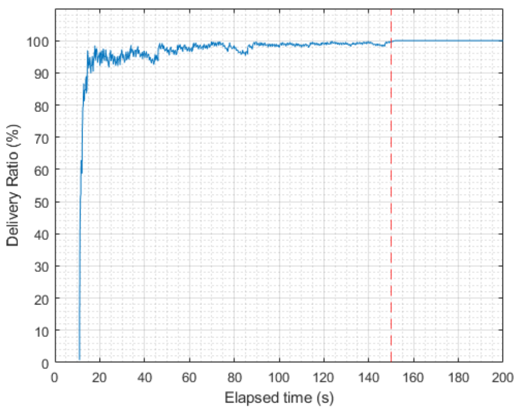


(b) M2C1.

Figure 5.22: Gateways elected in scenario C with control strategy 1.

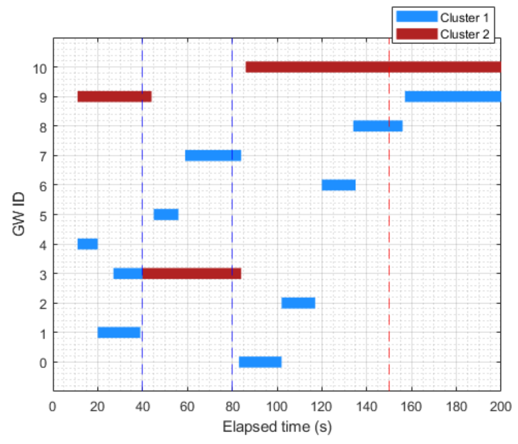


(a) Method 2.

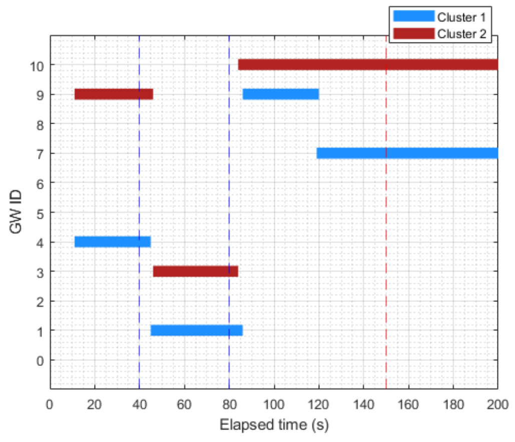


(b) Method 2.

Figure 5.23: Delivery ratio in scenario C with control strategy 2.



(a) Method 1.



(b) Method 2.

Figure 5.24: Gateways elected in scenario C with control strategy 2.

Parameter	M1C1	M2C1	M1C2	M2C2
Simulation events	783167	764348	753482	742298
Redundant overhead	210 pkts	137 pkts	33 pkts	14 pkts
Mean delay	2.6143 s	3.2645 s	1.4379 s	1.4205 s
Total data hops	4497	2949	2085	1606
GW elections	44	39	12	7
Battery difference	4%	24%	12%	35%

Table 5.6: Results for scenario C.

In Figure 5.24a we can also observe that there are two elected GW in cluster 1 from around the 30 to the 40 seconds and between the new election of GWs exists a gap of a few seconds without an GW elected. Such occurs because the USV 3 moves around that period of time, changing its centrality that is used to break the tie for having the same battery of other USVs, and this information can take longer to arrive to other USVs than the time that the control strategy takes to elect a new GW. The variation of the parameters of this second control strategy is evaluated on the next section.

Overall, the second control strategy has a much better performance than the first one, providing a more stable election of the GW and, consequently, a more stable network. Between the two methods, the second presents an overall better performance due to its additional parameters that lead to a better proposal of GWs for election.

5.3.4 Evaluation During a Monitoring Task

The evaluation of this section is made considering the aquatic scenario D, which represents a more complex and realistic monitoring scenario. Table 5.7 presents the configured parameters that were not yet described in Table 5.2.

General	
sim-time-limit	1000 *s*
numNodes	12
Application module	
endGeneratingMsg	850 *s*
Neighboring module	
EnergyStart	450 *J*
BS_position	(400, 25)

Table 5.7: Extra parameters for the configuration of the simulator in scenario D.

Many tests with different parameter values were performed during the evaluation of the developed work. This section presents the results obtained by using the second GW election method and the second control strategy. The parameters of the second and third control strategies were changed throughout several cases as presented in Table 5.8 in order to evaluate the implemented work. A first case aims to show the results of the behaviour when the parameters of the control strategies are too low, two demonstrate the behaviour of the control strategies with intermediate parameter values, and a final case has parameters whose values are too high. All these cases were repeated 10 times and their results are displayed in Table 5.9 with their average value and a 95% confidence interval. The following figures are from one of the 10 repetitions, the blue dotted lines represent the time where the clusters have a new formation, as presented previously in Figure 5.12, and the red dotted line represents the time the USVs stop generating messages.

Parameter	**Case 1**	**Case 2**	**Case 3**	**Case 4**
control_count	1	6	10	15
control_time	2 *s*	2 *s*	2 *s*	2 *s*
control_countP	3	5	20	60
control_timeP	1 *s*	1 *s*	1 *s*	1 *s*

Table 5.8: Different control strategies parameterization.

Case 1

For this case, the chosen values are low, considering that the beacons, from which the USVs update their view of the network, are only sent by each second. Having a control of 1 (*control_-count*) time plus 2 (*control_time*) seconds gives a very low window of time for the USV to rectify its proposed GW, ending on premature GW elections. The delivery ratio presented in Figure 5.25 shows that the test ended without the delivery of all the generated data messages, a situation that was frequent on several repetitions of this case. This occurred because, at the end, there was no consensus on the elected GW.

In order to be an active GW in the cluster and to attain a successful delivery of the data messages, the USVs of the cluster must elect the same GW and this GW must also elects itself. During this case, many USVs elected a different GW. Such difference led to the lack of a GW from the 679 seconds to the 758 seconds and after the 774 seconds in cluster 1, as shown in Figure 5.26. Also, the existence of more than one GW in the same cluster has occurred. Such happened from the 158 seconds to the 223 seconds, where on the first cluster both USVs 1 and 2 remained as GW. Another problem is that, even with the existence of a recognized GW in the cluster, with USVs in the path that elected a different GW, can lead to the forward of the data messages in the wrong direction, which can congest their delivery to the actual GW.

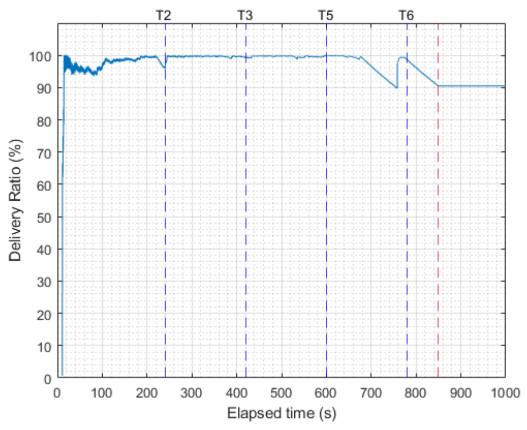


Figure 5.25: Delivery ratio in scenario D case 1.

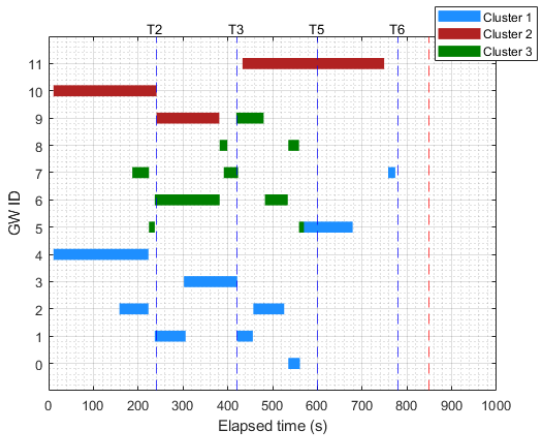


Figure 5.26: Gateways elected in scenario D case 1.

Case 2

For this case, a higher set of values for the parameters were chosen, but where the performance of the implemented work remained stable. In Figure 5.27 we can observe that the successful delivery of the data starts quickly after the start of the simulation. A slight drop around the 30 seconds occurs as 10 USVs deployed together change into their first formation (Figure 5.12b), which changes the paths from the USVs to the GW. Throughout the rest of the simulation, the delivery ratio remains near the 100%, except for an attenuation of around 5% at the 720 seconds. This attenuation is due to the fact that, between the 703 and the 738 seconds, there is not a consensus elected GW by the cluster, as shown in Figure 5.28. This delay on the election of the GW happens because, at the 780 seconds, the scenario presents a new formation, that it starts before as the USVs move previously to reach the new formation at that time. Consequently, the changes on the clusters happen some seconds before, which was the case here. As they had a new element on the cluster (USV 11), some delay was introduced on the election of the new GW.

From Figure 5.28, we can observe that the control strategy has a good behaviour, even though that the average delivery ratio at the end of the 10 simulations presented in Table 5.9 is not 100%. As the USVs stop generating data at the 850 seconds and keep their formation, they stop spending so much energy or changing their position relative to their neighborhood, consequently not changing their rank. So if a wrong election occurs within one of the USVs at this stage, it can stay unchanged as its rank is still 10% better than others, which can result on it sending the data messages on the wrong direction. Even so, such situation has a low probability of having a great impact on the network, as the mean end delivery ratio is 97.90% with a small confidence interval.

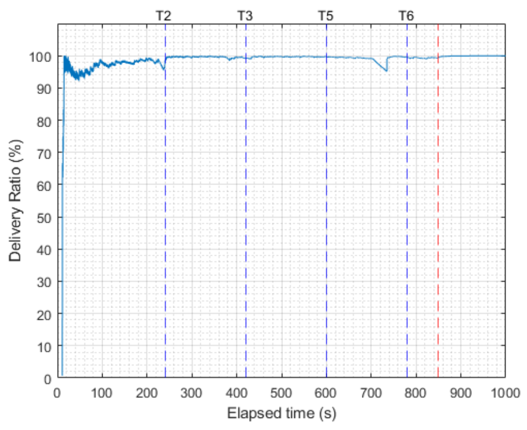


Figure 5.27: Delivery ratio in scenario D case 2.

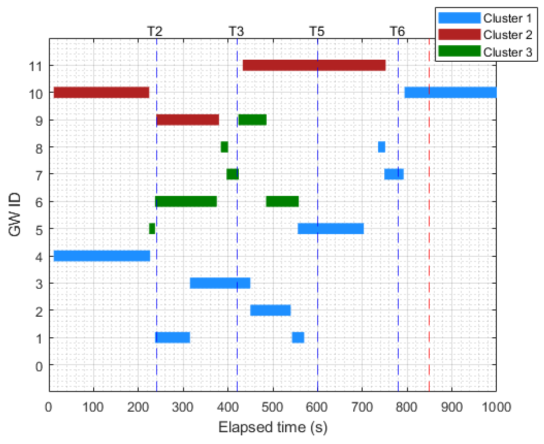


Figure 5.28: Gateways elected in scenario D case 2.

Case 3

For this case, higher values for the parameters were chosen, but in which the performance of the implemented work remained acceptable. Figure 5.29 shows that, initially, the delivery ratio takes some time before achieving around 90% of success. Such delay is because, at the beginning, the network has major changes, since the deployed USVs perform big changes on their positions before getting into the formation presented in Figure 5.12b. Also, now the parameters for the third control strategy are quite high, that is the strategy used when there has not been a GW elected or the cluster suffered changes. Therefore, the first election on the first cluster only gets successful at 73 seconds of run time, as shown in Figure 5.30. However, it does not go above 90% until after the 240 seconds because some USVs end up not electing a GW before that time. It was also observed that, because the parameter of the second control strategy is higher, it elects less times. These behaviours indicate that the parameters on both control strategies are close to being too high, specially when the cluster presents high mobility and formation changes, as it is the case at the beginning of this scenario. The remaining time the overall behaviour remains good and stable, having a mean end delivery ratio on the 10 repetitions of 99.73%, with a considerably small confidence interval, as presented in Table 5.9.

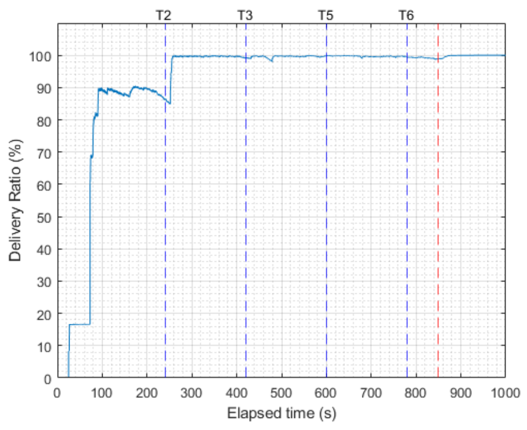


Figure 5.29: Delivery ratio in scenario D case 3.

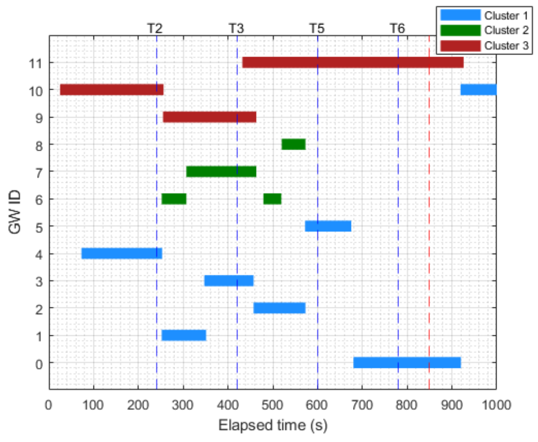


Figure 5.30: Gateways elected in scenario D case 3.

Case 4

This case has the control strategies parameterized with higher values when compared to the previous ones. As expected from the results of the last case, the delivery ratio shown in Figure 5.31 takes a long time before achieving values near the 100%. This happens because, having control parameters that are very high, delay the election of the GWs, specially in clusters with a higher number of USVs, and where the network has more changes during time. Therefore, if an USV stays too much time checking the proposed GW, it can end up changing the proposed GW before even electing one. Figure 5.32 shows that the GW on the biggest cluster at the beginning only elects itself at the 162 seconds, and its election by the other USVs is made with a high difference of time in between. It was also observed that the majority of the GW elections are made by the third control strategy, as the second takes a large time to elect a GW that, eventually, the cluster suffers changes and it jumps to the third strategy. Also, in Figure 5.32, we can observe that the USV 9 after the 420 seconds, although being in a new cluster, has still remained itself as GW because it has not elected a new one, that only elects at the 558 seconds of simulation run time.

Having very high parameters can introduce another problem that, although not presented on the majority of the simulations we run under this case, it happened in 2 out of the 10 tests of this case. This problem is an USV staying stuck as GW because there is no consensus on electing a new GW for too much time. This results on the drainage of the battery of this USV. Besides the appointed issues, under higher values, there is actually a higher probability of ending the simulation with 100% of delivery ratio, as there are less changes on the elected GW, but at a cost of a higher probability of draining the USV's battery.

Having different parameters produce different behaviours of the GW election. From Table 5.9 we can observe that low control values result on a higher number of GW elections, having more

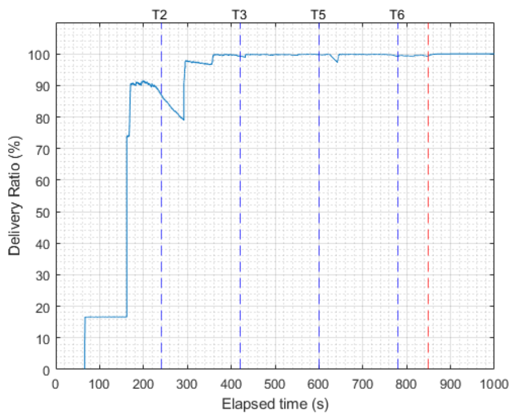


Figure 5.31: Delivery ratio in scenario D case 4.

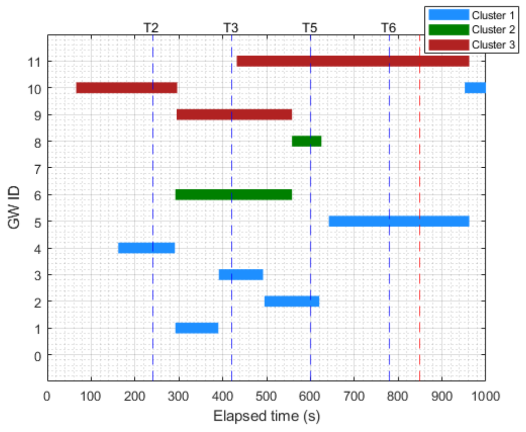


Figure 5.32: Gateways elected in scenario D case 4.

that one GW in the cluster. Such situation adds confusion within the network and results on a higher number of hops, as data messages can get on back and forward between USVs until they finally elect the same GW. Having more data messages going through the network also increases the redundant overhead, as the loss of acknowledgment packets increases. For higher values there is a higher delivery ratio at the end of the simulation, but also a higher delay on the delivery of the data messages and a higher unevenness among the USV's battery. An overall better behaviour was observed with the parameters used in case 2, where results are fair and have a far better confidence interval than the other cases, which indicate the more stable behaviour of this set of values for the parameters of the control strategies.

Parameter	**Case 1**	**Case 2**	**Case 3**	**Case 4**
End delivery ratio	95.11 ± 3.40%	97.90 ± 2.81%	99.73 ± 0.31%	99.92 ± 0.15%
Redundant overhead	270 ± 80 pkts	109 ± 32 pkts	134 ± 23 pkts	459 ± 376 pkts
Mean delay	5.29 ± 1.54 s	3.57 ± 0.34 s	5.84 ± 0.81 s	16.60 ± 4.98 s
Total data hops	17717 ± 6289	11906 ± 862	14407 ± 2648	11588 ± 1809
GW elections	22 ± 1	18 ± 1	14 ± 1	11 ± 1
Battery difference	50 ± 4%	65 ± 9%	62 ± 8%	75 ± 9%

Table 5.9: Results for scenario D with 10 repetitions.

Altogether, tuning the control strategies with good values for their parameters takes some tries, as it was observed from several tests that, having intermediate values can lead to some lack of consensus within the cluster on electing a GW. This occurs due to an asynchronous election, that is, the USVs end up electing GWs at different periods of time that are separated enough to lead to the election of different USVs as GWs. Therefore, having a simulator available to evaluate the parameters before the implementation on a real environment is a useful resource.

5.4 Chapter Considerations

This chapter depicted the hardware and software used in this section. It was presented the validation of the proposed simulator by comparing results on different scenarios, with different protocols, and with results from simulations performed by [3]. The implemented GW election methods and control strategies were evaluated through a set of different scenarios, first comparing the methods and control strategies among each other, and then by evaluating the proposed GW election method 2 with the control strategies 2 and 3 through several cases under a scenario that simulated a real monitoring task. The evaluations allowed to see the behaviour of each GW election method and control strategy regarding the delivery ratio, the GWs elected within the cluster throughout the time, as well as other important parameters for the evaluation of the behaviour of the presented work. The next chapter presents the conclusions of the work developed along this dissertation.

Chapter 6

Conclusions and Future Work

Monitoring an aquatic environment requires several swarms of USVs with constant communication within each other, that will be collecting data from the sensors to be forwarded to the station on land. Therefore, the USVs must be able to self-organize and elect the cluster's GW, which will be responsible for forwarding the gathered data to the BS. Testing network mechanisms, such as GW election methods and strategies, can be very costly and require a lot of time and resources; therefore, having a simulation platform is a handy resource to evaluate these network mechanisms before implementing them on real world scenarios.

The main goal of this dissertation was to study mechanisms of communication on self-adaptive multi-cluster of aquatic USVs environments, with clusters being formed and changed dynamically, through the implementation of gateway election methods and strategies, and develop a simulation platform capable of evaluating the implemented work.

A simulator capable of running mobile scenarios of swarms of drones in aquatic sensing environments was developed. This simulator presents a modular architecture, based on a DTN, where new network mechanisms can be straightforwardly implemented. The evaluation of the simulator was performed by comparing its behaviour with another similar, but low-scale, simulator. From these evaluations, we concluded that the simulator presents a good behaviour, as expected.

In order for a cluster to elect a GW, two GW election methods and three control strategies were implemented and evaluated. The second method takes into consideration more metrics besides the energy of the USVs, such as the link quality. The second control strategy, compared to the first one, has a higher control on the GW election flux from the methods, and the third control strategy is triggered when there has not been a GW election or when there is a change on the cluster's formation. It was observed that, with the first method, the USVs end the tasks with a similar battery level, while with the second method, they present uneven levels of battery. Even though, the second method has a better overall performance, as it achieves a lower mean delay and a better ratio on the packet delivery, with a lower number of hops of the data packets. The control strategies showed to be crucial for the network behaviour. With the first control strategy, a high number of GWs were elected throughout the simulations, leading to a more unstable network, while the second strategy provided more stability, reducing the total number of GW elections, and preventing the wrong elections of GWs within the cluster.

Finally, it was evaluated the second GW election method with the second and third control strategies by varying their parameters, during an aquatic monitoring task. It was observed that evaluating the parameters previously on a simulator helps enhancing the performance of the network. Parameters with values that are too low or too high can lead to undesired results, while intermediate values give more desired outcomes with a better confidence interval.

In summary, this work concludes that a simulation platform is a valid resource to implement and analyse network mechanisms in aquatic surface environments. Also, it concludes that GW elections performed through a passive mode based on several metrics, besides energy and centrality, but also of link quality, are a good investment to apply in aquatic monitoring environments with several swarms of drones.

6.1 Future Work

Considering that this dissertation is the starting point on the development of the presented simulator, many features can be improved and added. Also, there are still several elements that need to be improved on the GW election methods and control strategies. Noteworthy:

- The addition of more network interfaces, namely long range communication as, for example, LoRa;
- The improvement of the simulator's physical layer, in order to approximate even more the simulation environment to the reality. A more accurate physical layer can be implemented from results obtained by more real communication tests in aquatic environments;
- Implement reactive mobility in the simulator, so that scenarios where the USVs change their position as a reaction to a certain situation can occur within the simulation run time;
- Evaluate the implemented GW election methods and control strategies on longer monitoring scenarios with more changes of positions of the USVs within the cluster;
- Study mathematically the rank equation that elects the gateway, analysing the impact of the used metrics on this equation throughout several different aquatic scenarios, and understand if additional adjustments can lead to a better performance;
- Evaluate and study new approaches to improve the consensus and even synchronized GW elections in the cluster by, for example, passing additional information along the periodic beacon that is spread in the network;
- Implement and evaluate the proposed approaches in a real aquatic environment.

Bibliography

[1] R. Almeida, R. Oliveira, M. Luís, C. Senna, and S. Sargento. *A multi-technology communication platform for urban mobile sensing.* Sensors, 18(4), 2018.

[2] G. Pessoa. *Content Distribution in Vehicular Networks using Delay-Tolerant Communication Mechanisms.* Master's thesis, University of Aveiro, 2015.

[3] D. Sousa, M. Luís, S. Sargento, and A. Pereira. *An aquatic mobile sensing USV swarm with a link quality-based delay tolerant network.* Sensors, Vol. 18, No. 10, pp. 3440 - 3440, October, 2018.

[4] K. Fall and S. Farrell. *DTN: an architectural retrospective.* IEEE Journal on Selected areas in communications, 26(5), 2008.

[5] A. Vahdat and D. Becker. *Epidemic routing for partially-connected ad hoc networks.* http://issg.cs.duke.edu/epidemic/epidemic.pdf, 2000. Last accessed: 2019-10-18.

[6] B. M. Khan, R. Bilal , R. Young. *Fuzzy-TOPSIS based Cluster Head selection in mobile wireless sensor networks.* Journal of Electrical Systems and Information Technology, Volume 5, pp.928–943, 2018.

[7] W. Heinzelman, A. Chandrakasan and H. Balakrishnan. *Energy-efficient communication protocol for wireless microsensor networks.* In Proc. of the 33rd Annual Hawaii International Conference on System Sciences (HICSS), pp. 3005 – 3014, Maui, HI, Jan. 2000 .

[8] D. Moura, L. Guardalben, M. Luís, and S. Sargento. *A drone-quality delay tolerant routing approach for aquatic drones scenarios.* In 2017 IEEE Globecom Workshops (GC Wkshps), pages 1-7, Dec 2017.

[9] R. Oliveira, M. Luís, S. Sargento. *On the Performance of Social-based and Location-aware Forwarding Strategies in Urban Vehicular Networks.* Ad Hoc Networks, Vol. 93, No. 10, pp. 101925 - 101925, October, 2019.

[10] S. Burleigh, A. Hooke, L. Torgerson, K. Fall, V. Cerf, B. Durst, K. Scott, and H. Weiss. *Delay-tolerant networking: an approach to interplanetary internet.* IEEE Communications Magazine, 41(6):128–136, June 2003.

[11] F. Warthman et al. *Delay-and disruption-tolerant networks (dtns).* A Tutorial. V..0, Interplanetary Internet Special Interest Group, 2012.

[12] K. Scott and S. Burleigh. *Bundle protocol specification.* Technical report, 2007. http://www.rfc-editor.org/rfc/rfc5050.txt. Last accessed: 2019-10-14.

[13] F. Warthman and W. Associsatires. *Delay and Disruption-Tolerant Networks (DTNs) A Tutorial.* Based on Technology Developed by the DTN Research Group (DTN-RG), Version, 3:1-35, 2015.

[14] M. Demmer, E. Brewer, K. Fall, S. Jain, M. Densmore, and R. Patra. *Implementing Delay Tolerant Networking, Intel Research.* Berkeley, Technical Report, IRB-TR-04-020, December, 2004.

[15] S. Schildt, J. Morgenroth, W.-B. Pöttner, and L. Wolf. *Ibr-dtn: A lightweight, modular and highly portable bundle protocol implementation.* Electronic Communications of the EASST, 37, 2011.

[16] G. Pessoa, R. Dias, T. Condeixa, J. Azevedo, L. Guardalben, and S. Sargento. Content distribution emulation for vehicular networks. In *2017 Wireless Days*, pages 208–211, March 2017.

[17] N. Baccour, A. Koubâa, M. B. Jamâa, H. Youssef, M. Zuniga, and M. Alves. *A comparative simulation study of link quality estimators in wireless sensor networks.* In 2009 IEEE International Symposium on Modeling, Analysis Simulation of Computer and Telecommunication Systems, pages 1–10, Sept 2009.

[18] A. Woo and D. Culler. *Evaluation of efficient link reliability estimators for low-power wireless networks.* EECS Department, University of California, Berkeley, Tech. Rep. UCB/CSD-03-1270, 2003.

[19] M. Senel, K. Chintalapudi, D. Lal, A. Keshavarzian, and E. J. Coyle. *A kalman filter based link quality estimation scheme for wireless sensor networks.* IEEE Global Telecommunications Conference (GLOBECOM '07), 2007.

[20] J. Silva, B. Krishnamachari, F. Boavida (Eds.). *Wireless Sensor Networks.* In 7th European Conference, EWSN 2010, Coimbra, Portugal, pages 241-244, Frebruary 2010.

[21] D. Couto, D. Aguayo, J. Bicket, and R. Morris. *A high throughput path metric for multi-hop wireless routing.* MobiCom '03: Proceedings of the 9th annual international conference on Mobile computing and networking. New York, NY, USA: ACM, 2003, pp. 134–146.

[22] R. Fonseca, O. Gnawali, K. Jamieson, and P. Levis. *Four bit wireless link estimation.* Proceedings of the Sixth Workshop on Hot Topics in Networks (HotNets VI), 2007.

[23] N. Baccour, A. Koubâa, H. Youssef, M. Ben Jamâa, D. do Rosário, M. Alves, and L. B. Becker. *F-LQE: A Fuzzy Link Quality Estimator for Wireless Sensor Networks.* In Wireless Sensor Networks, pages 240–255. Springer Berlin Heidelberg, 2010.

[24] W. Sun, W. Lu, Q. Li, L. Chen, D. Mu, and X. Yuan. *Wnn-lqe: Wavelet-neural-network-based link quality estimation for smart grid wsns.* IEEE Access, 5:12788–12797, 2017.

[25] D. Sousa. *A communication network for sensing by a self-adaptive team of aquatic drones.* Master's thesis, University of Aveiro, 2018.

[26] T. Spyropoulos, K. Psounis, and C. Raghavendra. *Singlecopy routing in intermittently connected mobile networks.* In Sensor and Ad Hoc Communications and Networks, 2004. IEEE SECON 2004. 2004 First Annual IEEE Communications Society Conference on, pages 235-244. IEEE, 2004.

[27] P. Pereira, A. Casaca, J. Rodrigues, V. Soares, J. Triay, and C. Cervello-Pastor. *From Delay-Tolerant Networks to Vehicular Delay-Tolerant Networks.* IEEE Communications Surveys Tutorials, 14(4):1166–1182, Fourth 2012.

[28] T. Almeida. *Delay tolerant network for navy scenarios: quality based approach.* Master's thesis, University of Aveiro, 2015.

[29] N. Aierken, R. Gagliardi, L. Mostarda, Z. Ullah. *RUHEED- Rotated Unequal Clustering Algorithm For Wireless Sensor Networks.* 29th International Conference on Advanced Information Networking and Applications Workshops, 2015.

[30] W. Heinzelman, A. Chandrakasan, and H. Balakrishnan. *Energy-Efficient Communication Protocol for Wireless Microsensor Networks.* Published in the Proceedings of the Hawaii International Conference on System Sciences, January 4-7, 2000, Maui, Hawaii.

[31] O. Younis and S. Fahmy. *Distributed Clustering in Ad-hoc Sensor Networks: A Hybrid, Energy-Efficient Approach.* Published in: IEEE INFOCOM 2004.

[32] D. Wei, Y. Jin, S. Vural, K. Moessner, R. Tafazolli. *An EnergyEfficient Clustering Solution for Wireless Sensor Networks.* IEEE Trans Wirel Commun 10(11):3973–3983. 2011.

[33] N. Aslam, W. Phillips, W.Robertson and S. Sivakumar. *A multicriterion optimization technique for energy efficient cluster formation in wireless sensor networks.* Inf Fusion 12(3):202–212. 2011.

[34] M. Negnevitsky. *Artificial intelligence: a guide to intelligent systems.* 1st edn. Addison-Wesley Longman Publishing Co., Inc., Boston, 2005.

[35] S. Murugaanandam and V. Ganapathy. *Reliability-Based Cluster Head Selection Methodology Using Fuzzy Logic for Performance Improvement in WSNs.* In IEEE Access (Volume: 7). July 2019.

[36] P. Mehra, M. Doja, B. Alam. *Fuzzy based enhanced cluster head selection (FBECS) for WSN.* Journal of King Saud University - Science, April 2018.

[37] P. Gupta, and A. Sharma. *Clustering-based heterogeneous optimized-HEED protocols for WSNs.* Soft Computing. April, 2019.

[38] M. Manhães, S. Scherer, M. Voss, L. Douat, and T. Rauschenbach. *Uuv simulator: A gazebo-based package for underwater intervention and multi-robot simulation.* In OCEANS 2016 MTS/IEEE Monterey, pages 1–8, Sept 2016.

[39] 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, ser. Simutools '09, Rome, Italy, Mar. 2009, pp. 55:1–55:10.

[40] P. Katkar, V. Ghorpade. *Comparative Study of Network Simulator: NS2 and NS3.* International Journal of Advanced Research in Computer Science and Software Engineering. Volume 6, Issue 3, March 2016 .

[41] J. Dede, A. Förster, E. Hernández, J. Herrera, K. Kuladinithi, V. Kuppusamy, P. Manzoni, A. bin Muslim, A. Udugama, Z. Vatandas. *Simulating Opportunistic Networks: Survey and Future Directions.* IEEE Communications Surveys & Tutorials, Volume: 20, Issue: 2, Second quarter 2018.

[42] M. Khan, H. Hasbullah, B. Nazir. *Recent open source wireless sensor network supporting simulators: A performance comparison.* Published in 2014 International Conference on Computer, Communications, and Control Technology (I4CT).

[43] N. Vastardis, K.Yang, S. Leng. *Socially-Aware Multi-phase Opportunistic Routing for Distributed Mobile Social Networks.* Wireless Pers Commun (2014) 79: 1343.

[44] A. Udugama, A. Forster, J. Dede, V. Kuppusamy and A. Bin Muslim. *OPS - An Opportunistic Networking Protocol Simulator for OMNeT++.* Proc. of the 4th OMNeT++ Community Summit, University of Bremen - Germany - September 7-8, 2017.

[45] R. Oliveira. *Forwarding Strategies for Opportunistic Data Gathering in Mobile IoT.* Master's thesis, University of Aveiro, 2018.

[46] H. Karvonen, C. Pomalaza-raez et al. *Experimental performance evaluation of ble 4 vs ble 5 in indoors and outdoors scenarios.* ACM, BODYNETS'17, Dalian, China, 09 2017, pp. 123–4567–24–567/08/06.

[47] L. M. F. Oliveira. *Vehicular networks: Ieee 802.11p analysis and integration into an heterogeneous wmn.* Master's thesis, Fac. de Eng. da Univ. do Porto, June 2012.

[48] U. Paul, R. Crepaldi, J. Lee, S. Lee, and R. H. Etkin. *Characterizing wifi link performance in open outdoor networks.* Proceedings of the 8th SECON 2011, Salt Lake City, UT, USA, 6 2011, pp. 251–259.

[49] J. Liu et al.. *Modeling neighbor discovery in bluetooth low energy networks.* IEEE Comm Letters, vol. 16, no. 9, p. 1439, Sep. 2012.

[50] J. Liu, C. Chen, Y. Ma, and Y. Xu. *Energy analysis of device discovery for bluetooth low energy.* VTC Fall, IEEE 78th, 09 2013, pp. 1–5.

[51] I. Gupta, D. Riordan, S. Sampalli. *Cluster-head election using fuzzy logic for wireless sensor networks.* Published in: 3rd Annual Communication Networks and Services Research Conference (CNSR'05). 2005.

[52] Ulrik Brandes. *A Faster Algorithm for Betweenness Centrality.* Journal of Mathematical Sociology. Volume 25, 2001 - Issue 2.

[53] K. Wehrke, M. Günes and J. Gross. *Modeling and Tools for Network Simulation.* Springer Science and Business Media, 2010.