A novel online functional testing methodology based on a fully distributed continuous monitoring approach applied to communicating systems

Jose Alfredo Alvarez Aldana

NNT : 2018SACLL005

THÈSE DE DOCTORAT
DE
L'UNIVERSITÉ PARIS-SACLAY
PRÉPARÉE À
TELECOM SUDPARIS

ÉCOLE DOCTORALE STIC
Intitulé complet de l'école doctorale

Spécialité de doctorat : Informatique

Par

Monsieur José Alfredo ALVAREZ ALDANA

Une Méthode de test fonctionnel en-ligne basée sur une approche de monitorage distribuée continue appliquée aux systèmes communicants

Thèse présentée et soutenue à Evry, le 28 septembre 2018

Composition du Jury :

M. Philippe DAGUE	Professeur, Université Paris-Sud	Président du Jury
M. Philippe JACQUET	Directeur de Recherche, Nokia Bell-Labs	Examinateur
M. Farid NAIT-ABDESSELAM	Professeur, Université Paris Descartes	Rapporteur
M. Hacène FOUCHAL	Professeur, Université de Reims Champagne	Rapporteur
M. Stephane MAAG	Professeur, Telecom SudParis	Directeur de thèse
Mme. Fatiha ZAIDI	Professeur, Universite Paris-Sud	Co-encadrant

Preface

Notation Conventions

General notation conventions adopted in the present work are summarized as follows:

- Scalar-valued quantities used in tables, are approximate values, which are truncated to two decimal digits. For the ease of the reading the approximate symbol ($\approx$) has been removed.
- The dimensions are omitted in the tables. The timeout is represented in milliseconds. The convergence time is represented in milliseconds. The complete time is represented in milliseconds. The observations are represented in number of nodes. The network size is represented in meters.

Publications and License Information

The present PhD thesis has produced the publications that were submitted and accepted by the corresponding conference entity. For the time being, there are some publications which still remain under review. The author declares that only personal contributions are used in this thesis.

- Alvarez, J., Maag, S., & Zaïdi, F. (2016). Manets monitoring with a distributed hybrid architecture. In *Network computing and applications (nca), 2016 ieee 15th international symposium on* (Pages 388–391). IEEE.
- Alvarez, J., Maag, S., & Zaïdi, F. (2017b). Monitoring dynamic mobile ad-hoc networks: A fully distributed hybrid architecture. In *31st IEEE international conference on advanced information networking and applications, AINA 2017, taipei, taiwan, march 27-29, 2017* (Pages 407–414). doi:10.1109/AINA.2017.74.
- Alvarez, J., Maag, S., & Zaïdi, F. (2017a). Dhymon: a continuous decentralized hybrid monitoring architecture for manets. *arXiv preprint arXiv:1712.01676*.
- Alvarez, J., Maag, S., & Zaïdi, F. (2018). A formal consensus-based distributed monitoring approach for manet. In *Computer networks (under review)*. ELSEVIER.

The present PhD thesis is written with TEX. The base template was taken from a Bitbucket repository from Tianyi Li (Li, 2016) from Université ParisTech, who kindly shared his sources files to others to utilize. The source files of this thesis are available in a public Github repository:

https://github.com/chepeftw/PhDThesis


```
@PhdThesis{Alvarez:2018,
  author = {Alvarez, Jose},
  title = {{D}ecentralized and {D}istributed {M}onitoring for {M}ANETs},
  school = {Telecom SudParis, Université Paris-Saclay},
  year = {2018},
  month = sept,
}
```

Interested readers can freely use or adapt the document structure, the title page, etc., to their own needs.

List of Figures

List of Tables

List of Theorems

Contents

1

Introduction

Mobile ad-hoc networks: description and challenges

MANETs represent an important area of network research since they constitute various problematics but at the same time, multiple applications (Kiess & Mauve, 2007), therefore many opportunities. According to the survey of Conti et al. (Conti & Giordano, 2014), the MANET paradigm emerged in the 1990s with the possibility for users to connect with each other through Bluetooth (IEEE 802.15.1) and the wireless LAN (IEEE 802.11). These specifications allowed devices to communicate directly with each other through a single-hop ad-hoc network. The obvious remark is that for this type of network, there was no need for infrastructure since the devices by itself were sufficient as long as if they were within their antennas range. From this, researchers extended the concept to build a more extensive network beyond the range of the antennas by allowing the network packets to go through more than one hop. This was done around the same time, and it brought much attention due to the flexibility it showed and the lack of network infrastructure. This new network paradigm allowed the users and nodes of a network to interchange their data and also, through the proper mechanisms, to relay Internet traffic. Therefore, it became a promising technology, which gained much attention together with the fourth-generation (4G) of wireless networks. This drove a lot of research interest around the 2000s as it can be mentioned for Basagni et al. (Basagni, Conti, Giordano, & Stojmenovic, 2004) and Chlamtac et al. (Chlamtac, Conti, & Liu, 2003).

Initially, the community focused on pure general purpose MANETs, meaning that the proposals tried to be generalized for all possible scenarios. This implies that the network was genuinely infrastructure-less. It also meant that there was no authority in charge, meaning it worked in a distributed way. Moreover, it also meant that there were not any specific applications, and the ideas

were to support the legacy network applications. Following this trend, there were initial efforts to adapt the traditional protocols from wired networks to these networks, but it was not the case given the dynamical nature of the networks. From this, there were significant efforts in working in the routing and forwarding capabilities of these networks. The more prominent protocols based on the IETF [1] were the Ad-hoc On-demand Distance Vector (AODV), Optimized Link State Routing (OLSR), Dynamic Source Routing (DSR) and Topology Broadcast based on Reverse Path Forwarding (TBRPF). Although the numerous proposals, there was not any single prominent protocol. All of these protocols had different pros and cons varying on different perspectives. Up to this day, there are still on-going debates for reactive protocols (e.g., AODV version 2) and proactive protocols (e.g., OLSR version 2) in the context of MANETs. Regardless of the initial interest and the potential applications, there was a missing motivation for more research works in this area. Two works by Conti et al. (Conti & Giordano, 2007b, 2007a) tried to tackle this question by analyzing the theory and the reality of MANETs. They concluded that the main reasons for the lack of interest and traction for MANETs were:

- Implementation, integration, and experimentation
- Simulation credibility
- Socio-economic motivations.

Regardless of this, these works also pointed out that there were novel ramifications from the MANETs like mesh networks, vehicular networks, opportunistic networks and sensor networks. In the recent years, there has been a regaining of attention to the overall context in MANETs due to the specialization of the approaches and the improvements in the previously mentioned reasons that somehow limited the research. Nowadays, there are more and easier ways to share, contribute and generate implementations and experiments through code repositories. There have been many studies trying to increase the credibility of the simulations by outlying recommendations and minimum requirements. Finally, due to the specialization of different applications, it has gained the needed socio-economic motivation.

The research up to this day agrees that there is a set of problematics that affects any proposal in this context. The most recurring problematics, as stated by Raza et al. (Raza, Aftab, Akbar, Ashraf, & Irfan, 2016), of MANETs are:

- Wireless properties
- Mobility
- Availability
- Limited resources

These problematics affect multiple aspects of communications inside a MANET. The wireless properties induce a different type of problems depending on the point of observation. The first is the range of communication. It is known that a wireless transmitter is not capable of infinite reachability. Therefore

[1] https://datatracker.ietf.org/wg/manet/charter/

each node has a physical limit for transmission and by transitivity the network as well. Another is the wireless link capacity, and this means that the radio channel is subject to many non-controlled external behaviors and changes. Therefore transmission can be affected by path harm, declining, intervention, and obstruction. Finally, the wireless medium itself, this means that transmissions are subject to a shared medium. This could incur in packet clashing which is quite regular in the wireless medium. Then we can turn to mobility, this means that the nodes are not in a fixed position and therefore this affects the possible path and links between the nodes. This can induce route changes and packet loss due to the mobility. The mobility also can create network fragmentation, which affects the connectivity of the nodes. This causes two or more subnetworks, and each can have their own set of information. Not only the information gets fragmented, but from the network perspective, it is difficult to determine that the network is fragmented indeed. The availability makes uncertainty about which nodes in the network we can rely on to send a message. This uncertainty creates the need for robust mechanisms that can withstand the availability of the nodes. Depending on the mechanisms in use, the availability can also be subject for route changes and packet loss. Finally, the limited resources force us to create efficient and optimized solutions which do not sacrifice resources as computing power or communications efforts. The resource limitation is usually reduced to energy consumption. Therefore, there is necessary to consider the number of packets generated and sent to the network, since this has a direct impact on the energy efficiency of the nodes. It is known that any communication module up to date is quite expensive regarding energy consumption. This, of course, is an on-going topic of research on a different research context. Although the MANET literature does not come aboard the how to make a node more energy efficient, it does try to make the proposals as efficient of possible for them to be energy-friendly.

Tackling network monitoring in MANETs

One prominent interest in any network is monitoring, and the MANET literature inherited this interest as well. Monitoring can be traced back to 1990s when the IETF released the RFC 1065 for SNMP (Simple Network Monitoring Protocol) for wired networks. We can agree that the research community probably started working on this topic back to the 1970s. Monitoring is described by Cormode (Cormode, 2013a) as "a number of observers making observations and wish to work together to compute a function of the combination of all their observations". The goal is that all observers (network nodes) compute a value $f(t)$ in a given instant of time t in a collaborative way. It is highlighted in the work of Conti et al. (Conti & Giordano, 2014), that monitoring is of high research interest and high market interest as well. It is known that when a network is deployed, there is always a need to know the state of the network. To know the state of the network allows the managers to improve, maintain, and debug the network at any moment. Also, as stated by Khan et al. (Khan, Peters, Sahinel, Pozo-Pardo, & Dang, 2018), monitoring is the foundation for the control loop for autonomous networks. Therefore, we can agree that the monitoring of a network is of great importance.

To monitor a MANET implies that we need to understand and overcome the challenges in place. We can look at the problems of monitoring from the mobility perspective. If a node triggers the monitoring and its packet is dropped or lost due to the wireless medium, then it will never complete

the task. It could be the case that the node itself is off-range from the rest of the network or the monitored value is in a node off-range from the network. If a node triggers the monitoring while the network is fragmented, it will not be able to reach all the nodes. Therefore this can cause partial and incomplete results for the monitoring purposes. Another problem is the analysis and interpretation of the data by the network. Given the mobility and the availability of the nodes, it can be difficult to compute a function among all nodes. Therefore each node can reply for itself, but aggregation is needed to provide an overall result. Another problem is that wired networks are known to utilize centralized monitoring approaches mostly. Therefore sometimes an attempt is made to tailor these existing centralized solutions to the MANET, but these solutions face many problems since these architectures were not designed to withstand multiple dynamic properties. These problems affect the monitoring of the network and the existing solutions. The central node solution is affected by all the possible problematics for MANETs, meaning that due to mobility it might not be reachable by all nodes. Therefore, it generates more traffic which translates to high resource consumption to all the network. Although, this does not guarantee that the node receives the message since the packet or communication could get lost due to the wireless medium. Even worst, the central node could go off-range or offline, and in such case, the whole network would be without any monitoring capabilities. Due to this problematics, there is a need to consider non-centralized approaches to monitor a MANET. This has derived the interest towards decentralized and distributed approaches. It is worth mentioning that decentralized and distributed approaches are not only tailored for MANETs or similar networks. These approaches have shown outstanding performance in dense wired networks, like the case for cloud operators.

Most of the known monitoring approaches in the literature state that accuracy is not reliable due to the dynamic properties of the MANET. A significant number of works in the literature tries to improve the overall performance of the approaches, by focusing on accuracy and convergence time. Network fragmentation has a harsh impact on the accuracy of a decentralized and distributed approach. Given that the information is spread through the network then if a subset of nodes of the network fragments, there is a probability that the information for the monitoring result is fragmented as well. Therefore, there have been many studies that try and have successfully increased and maintained the accuracy of the approaches, but most of the time the accuracy decreases with the number of nodes or density of the network. Referring to the work of Stingl et al. (Stingl, Gross, Saller, Kaune, & Steinmetz, 2012), they state that the non-functional requirements of a monitoring mechanism are:

i) Performance

ii) Costs

iii) Fairness

iv) Scalability

v) Robustness

vi) Stability.

When we refer to performance, we refer to the accuracy and convergence time of the delivered results.

The costs refer to the overhead by the communication or processing of the data. Fairness can be analyzed in respect of performance and cost. The scalability refers to the ability to work in large and dense networks as well as in small networks. The robustness deals with the behavior of external and unpredictable events. Moreover, the stability is the ability to address the erratic behavior of autonomous nodes. Based on these requirements, we can observe that there is an area for improvements in the decentralized and distributed context for monitoring MANETs.

In the decentralized literature, we could observe that the existing solutions rely on gossip-based or hierarchical-based approaches. Gossip-based approaches demonstrate their robustness and stability in dynamic scenarios and changing topologies. Nonetheless, depending on the scalability, the cost and performance can be impacted. On the other side, hierarchical approaches show an efficient performance, cost, and scalability, although the robustness and stability may decrease in dynamic scenarios. This shows that the two major categories perform very good under different characteristics, requirements, and constraints of a network. Therefore, we hypothesized that a more prominent algorithm could be derived from these two approaches for broader scenarios.

Looking at a survey from NASA (Goodloe & Pike, 2010), which looks at monitoring distributed systems, they state that in such kind of systems there is a need to monitor multiple points in the network, and then reach a consensus among the different information to have useful and processed data. This led us to believe that a distributed system combined with a robust consensus algorithm could provide a robust monitoring mechanism. From a different context, we observed a novel and similar solution being applied in a different domain, the blockchain. Therefore, we took this as an inspiration bedrock and from there we started to build a proposal that tackles the problematics for monitoring a MANET. A blockchain relies on different concepts, a distributed ledger, a consensus algorithm, cryptographic mechanisms and network protocol. Which somehow resonates with the needs we observe for monitoring a MANET.

Objectives and contributions

Based on the previous problematics and motivations from the literature, we realized the many open challenges and challenges waiting to be tackled. In this section, we outline the objectives and contributions achieved by this thesis. The work is divided into three main topics, and for each topic, we tried to provide the theoretical background and supporting experiments to prove the feasibility of the approaches. The contributions are listed following.

- Focused on decentralized monitoring, one of the main contributions of this thesis is the proposal of a hybrid-based approach for decentralized monitoring of mobile ad-hoc networks in dynamic contexts. We define an architecture combining gossip-based and hierarchical-based algorithms for query dissemination and data aggregation. We perform the gossip-based approach to disseminate the query and in the process to build a virtual hierarchical topology (VHT) for a limited time window. Once the query is disseminated through all the network, with the support of the VHT, a hierarchical-based aggregation takes place. The idea is to provide a strong aggregation structure

but minimizing the frangibility of the topology by making it valid only during a small window of time. To provide robustness and stability, we provide intermediary gossip-based mechanisms to complete the aggregation even in dynamic environments. Alongside, we define a monitoring protocol that aims at helping the decentralized monitoring process. We believe that many approaches propose exciting techniques for querying and aggregating the data, but from our knowledge, no ones propose any definition of a protocol easing the nodes communications. To define our monitoring protocol, we rely on the needs based on our hybrid algorithms and through JSON (RFC 7159 (Bray, 2014)), we characterize the structure. Our expectation is not just to provide a structure but also a mathematical background for further analysis and testing. This contribution is presented in Chapter 3.

- We considered that the proposal for decentralized monitoring was highly efficient, but there was room for improvements and more experimentation. Therefore we propose the introduction of a multi-root node approach enhancing the monitoring process and aiming to minimize network fragmentation. By defining several root nodes, we considerably improve the accuracy of the results. We have conducted an extensive campaign of experimentation to assess the results of our proposal. This contribution is presented in Chapter 3, and more specifically in Section 3.2.4. This enhancement proposed a two-root node approach for a start, but then it was tantalizing to increase the number of nodes, but the question then became how many root nodes do we need to increase? It was immediately tempting to say that the more, the better and following that logic the best case was to have all the nodes to be root nodes. This opened the question to consider a different approach.

- When considering a different approach where all nodes were root nodes, led us to consider a distributed approach. Most of the works in the monitoring literature do not define the proper structures to define the unambiguous background for their proposals. Therefore, we start with the proposal of a mathematical definition that models local results for a single node and global results for a group of nodes or the whole network. These local results are the responses to a query definition that is propagated to the network. Finally, the supporting definitions that manage these results in the local view and then integrated into a global view definition. These definitions help us to provide a strong foundation for our distributed monitoring architecture. This contribution can be found in Chapter 4, and more specifically in Section 4.2.

- The proposal of a distributed monitoring architecture that relies on multiple points of observations being the nodes. A node or an outside entity can trigger a query in the network, which is propagated. Based on our previous mathematical definitions, each node replies a local view of the node if applicable. The approach also provides a consensus mechanism that allows the architecture to aggregate and provides a more meaningful result by analyzing the data from a bigger perspective. This means that the consensus mechanisms elects a leader, which changes for every query, and the leader aggregates the local views into a global view of the network. The presence of this mechanism ensures that the data processing is genuinely distributed and a democratized process. Given the fact that once these nodes broadcast the global result, each participant node in the network can double-validate the result and therefore it provides a second layer for the validity of the data. All results are stored integrally to ensure the trust of the data. Finally, we provide an appropriated mechanism to free stored results to avoid resource

consumption problems. This contribution is shown in Chapter 4.

- Finally and our last topic we cover is network emulation. This contribution was a derivate from our desire to improve our experiments by making them as credible as possible. For this, we developed a network emulator built in-house which combines Amazon Web Services, Docker, and NS3 based on exciting ideas from the emulation literature. To provide a useful emulator, we then proceed to document it properly for other researchers to use and without any attempt to promote our emulator, the emulator gained some user on its own. We were able to realize about this through the forks, stars and watchers statistics from the Github repository. For the matter of this thesis, the emulator allowed us to run thousands of emulations varying parameters like the number of nodes, network size, node density, speed, mobility pattern and timeouts. It not only allowed us to run massive campaigns of experiments but also allowed us to do it efficiently, manageable and fast. As a result, we were able to generate data and make an in-depth analysis of it by looking at average, maximum, minimum and standard deviation values for accuracy, convergence time, number and size of packets, and other relevant and interesting measurements. This contribution can be found in Chapter 5, but it is used in Chapter 3 specifically in Section 3.3 and in Chapter 4 specifically in Section 4.3.

Summary and Outline of the Thesis

In this chapter, we provide a preliminary survey and abstract of the monitoring of mobile ad-hoc networks literature. We focused on the significant problematics which can be summarized to the dynamic nature of the networks since they are affected by multiple external parameters. We defined that for monitoring a network, it can be done with a centralized, decentralized or distributed approach. Although the centralized approach sounds more comfortable and efficient for wired networks, it is not the case when applied to a MANET. Therefore the decentralized and distributed approaches become an interesting and exciting alternative to achieve monitoring. We then propose a decentralized approach with a corresponding enhancement. Then through the conception of this enhancement, we proposed a new approach by going into the distributed domain. Finally, we achieved multiple promising experiments relying on a custom emulator that allowed us to test and measure the performance of our proposals.

The remaining of this thesis is as it follows. In Chapter 2, we propose a more detailed survey on the current state of the monitoring literature focused on MANETs. We highlight the achievements of the current approaches, but we also take note of the open questions, and we try to tackle them in this thesis. Then, in Chapter 3, we propose our first approach to monitor a MANET in a decentralized fashion trying to maximize accuracy and minimize convergence time. For this, we run multiple experiments and iterations to try to achieve an efficient proposal. Past this, we see the feasibility to propose a distributed approach, which is detailed in Chapter 4. We start by defining multiple mathematical definitions and structures to support our approach. Followed by the proper implementation and experiments to achieve a proper study of our approach. In Chapter 5, we present the tools and implementations used in this thesis. We start by outlying the architecture of our emulator since it played a primary role in our experiments. Then we present the implementations and considerations that we did for the implementations of our

proposals since past the mathematical background we also faced implementation challenges that we had to overcome. Finally in Chapter 6, we conclude with some remarks about our work and some future directions of our research.

2

State of the art

Contents

The literature has a wide range of works and proposals for the monitoring context. These proposals are applied to different networks. Based on the work of Battat et al. (Battat, Seba, & Kheddouci, 2014), it can be said that monitoring can be classified into centralized, decentralized or distributed monitoring. This classification is exemplified in Figure 2.1. The centralized type is the case where all the information is sent, processed and handled by a single point or node in the network. This is an excellent solution for small to big wired networks. Although, if the number of nodes is too high it could generate too much network traffic and therefore other solutions might be desirable. Once again, the centralized solutions seem to suffice a fair amount of percentage of the existing cases. If we consider wireless networks, it becomes evident that a centralized approach might not suffice most of the situations. It might be tempting to apply a centralized solution to a wireless network due to the extensive studies that there exist. In reality, in some situations, this could work, but there is a need for assumptions or requirements for the nodes to make it work. Then, if we want to design a generalized or more general approach, centralized approaches are not the best solution for wireless networks. The complexity of monitoring a network can quickly increase depending on the type of the network.

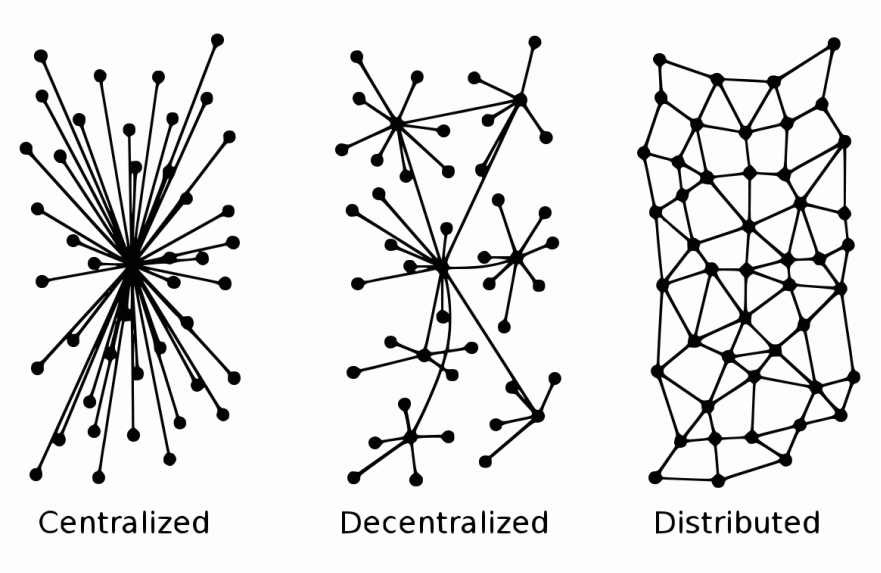


Figure 2.1 – Centralized, Decentralized and Distributed networks types exemplification

When we look at the specific case for MANETs, there are new considerations to make, like for example the mobility and the resource limitations. These considerations play a significant role in designing efficient monitoring approaches for wireless networks and more specifically for MANETs. The purpose of this chapter is to review the existing literature for decentralized and distributed monitoring for MANETs. It is highlighted in the work of Conti et al. (Conti & Giordano, 2014), that monitoring is of high research interest and high market interest as well. The existing literature contains a comprehensive set of exciting proposals. Nonetheless, there are open questions and areas for improvements that we herein try to tackle.

This chapter is divided as follows: In Section 2.1, we go through the proposals for decentralized approaches for monitoring MANETs. There are multiple subcategories, but we focus mainly on approaches relying on gossip and hierarchical algorithms. Each one of them has interesting proposals, but they have notorious difficulties in many scenarios. Then, in Section 2.2, we detail in depth the proposal for distributed approaches for monitoring MANETs. Some of the approaches rely on decentralized approaches, but then they provide the necessary mechanism to distribute the collected information. This could be simple and efficient, but it can have different drawbacks. Our proposal described in the following chapters, specifically the proposal of Chapter 4, is inspired by the blockchain technology. Therefore we intended to also look into the existing literature on blockchain related to our proposal. Finally, in Section 2.3, we describe and expand the open lines of research from the existing literature. Alongside this, we highlight the importance of our approach and how it tackles each one of the existing open questions.

2.1 Decentralized Monitoring for MANETs

The decentralized approach deals with networks where there is no fixed coordinator or centralizer entity. This implies that the network by itself needs to achieve a global view of a property of the network.

The preferred approach is to get the global view of a property of the network, and then disseminate it through the entire network to assure that is available to all the network. Referring to the works of Stingl et al. (Stingl et al., 2012; Stingl, Groß, & Saller, 2013), in the decentralized monitoring context we can mention three prominent categories:

a) Gossip-based approaches

b) Hierarchical-based approaches

c) Hybrid-based approaches

2.1.1 Gossip-based approaches

Gossip algorithms rely on selecting, from a set of reachable nodes, a random or a specific node (depending on the algorithm) to forward the data packet. There are multiple types of gossip techniques, the work of Chakchouk et al. (Chakchouk, 2015) provides a more detailed description of them. Although in this survey, the primary focus is on routing protocols, it explains the different techniques, properties and goals of gossip-based protocols. These protocols are designed using different properties according to their purpose. For example, energy efficient gossip-based protocols interchange information about their batteries to be more efficient. Another example is probabilistic gossip-based protocols that use the probabilities of meeting a node to their advantage. A general assumption in this kind of protocols is that the nodes require to be synchronized. Epidemic algorithms are subcategorization of gossip algorithms, which try to forward the packet not only to one but multiple nodes. Flooding is the most common and straightforward algorithm for epidemic algorithms. Gossip-based monitoring algorithms have the advantage of being highly stable and perform better in increasing dynamic networks. However, it may generate lots of traffic and, under specific scenarios, require more time to compute a value.

In the gossip-based categorization for monitoring, we can discuss Mobi-G (Stingl, Retz, Richerzhagen, Gross, & Steinmetz, 2014) which is inspired by Gossipico (Van De Bovenkamp, Kuipers, & Van Mieghem, 2012) for static networks. Gossipico is an algorithm to calculate the average, the sum or the count of node values in a vast dynamic network. The primary focus is to count nodes in networks since this information can be useful for the performance of routing protocols, topology parameters or merely to determine the size of the network. The foundation of the algorithm is through two procedures: count and beacon. The count procedure is the exchange of messages with other nodes using a gossip algorithm to disseminate this information through all the network. This does mean that all the nodes exchange all of the information they have until all the network has all the information. The beacon part is a mechanism to select a beacon node to guide the information collection messages towards each other. In the beginning, every node is a beacon, but they all compete with each other to make sure that eventually there is only one beacon node. The combination of two mechanisms, count, and beacon, provides the nodes of the network counting efficiently and quickly. This algorithm was tested in static networks. To test the robustness, they manually killed some nodes during the test to support the evidence that the algorithm is capable of dealing with dynamic scenarios. Although this proposal showed excellent performance for static networks, it was not designed or tested in wireless and mobile

networks. Therefore, Mobi-G was proposed for ad-hoc networks. It relies on the same principles and procedures. This means that it has a count procedure and a beacon procedure. The count is intended to share information and beacon to work as a facilitator in this process. It is designed for urban outdoor areas with a focus on pedestrian that moves around by foot. The idea is to create the global view of an attribute, which is built ideally incorporating all the nodes in the network. This global view is disseminated to all the nodes in the network to inform the current system state. It can provide accurate results even for fluctuating attributes. It also can reduce the communication cost. It does not suffer from long range connectivity, but the accuracy decreases for an increasing spatial network size. Based on the results, for small networks, the accuracy ranges from 98% to 94%, but for large or dense networks it decreases below 90%. These results show what we could consider a reasonable accuracy. On the other side, the convergence time can range from 60 seconds to 120 seconds, if not more, depending on some configuration parameters of their approach. This is not a good value when it is compared to other approaches. Although the approach is slow, the authors claim that the accuracy is maintained and the network traffic generated is low.

We can also mention the work of Guerrieri et al. (Guerrieri, Carreras, De Pellegrini, Miorandi, & Montresor, 2010), where they propose a distributed monitoring for global parameters but focused on delay-tolerant networks (DTN). They present four small algorithms and analyze them. These algorithms are an adaptation of popularly known aggregation approaches for networks. These algorithms are pairwise averaging (Jelasity, Montresor, & Babaoglu, 2005) and population protocol (Aspnes & Ruppert, 2009). The pairwise averaging algorithm is gossip/epidemic protocol. The main contribution of this work is that they designed an aggregation procedure on top of a gossip algorithm. They were inspired by the works done on epidemic communication for sending database updates. Before this work, aggregation was done relying on a hierarchy, and gossip algorithms were used only for routing. The population protocol states that computation is done between two mobile agents or nodes interacting. Each participant node is programmed as a finite state machine (FSM). Therefore, the interaction between the pair of nodes causes them to change the state of their FSM. These protocols are inspired by the models of interacting molecules in theoretical chemistry. The four algorithms that they proposed are based on the previously mentioned approaches. The first is an adaption of pairwise averaging for DTN relying on the population principle. The second is named population, which is a population-based algorithm. The third is named population-c, which is an improvement of the previous one, where the memory of the interactions is kept to form clusters. Finally, the fourth is named 2-phases, which has a further procedure to speed up the overall result relying on the probability of meeting other nodes. They evaluate their proposed algorithms evaluating the convergence time and accuracy. The accuracy is promising but at a high cost of time. The convergence time can be 600 seconds, and according to the study, it can decrease to 100 seconds if the network has more nodes. From another perspective, when they look at the accuracy of population-c and 2-phases, the accuracy of population-c is about 75% after 40,000 seconds and for 2-phases is 100% after 20,000 seconds. It can be said that the approaches are interesting, but the convergence time is extremely high.

Another interesting work is the one from Wuhib et al. (Wuhib, Dam, Stadler, & Clem, 2009), they propose a gossip protocol for continuous monitoring for network aggregates named G-GAP (stands for Gossip-bassed Generic Aggregation Protocol). The idea of this work is to asses the use of a gossip protocol for real-time aggregation in support of network monitoring. They argue that gossip protocols

have substantial advantages over hierarchical approaches and therefore the motivation for this work. The first is simplicity, gossip protocols do not need any predefined or maintain an on-going structure to operate. The second is the data location, in hierarchical approaches is focused on a root node meanwhile on gossip protocols it can be distributed by the same approach if needed but is not bound to a specific node. The third is failure handling, given that gossip protocols work in an unstructured approach, therefore it can cause mass loss. This means that if the node that is handling the packet fails, then the whole transmission is lost. Unless there are supporting mechanisms to avoid this of course. The protocol relies on the concept of push-synopses protocol from Kempe et al. (Kempe, Dobra, & Gehrke, 2003). Based on this concept which is prone to the mass loss problem, they extend it and make it more robust to handle these problems. The proposed protocol is designed to form part of a management architecture. This architecture communicates via an overlay network, where the goals are: a) accuracy b) controllability c) scalability d) and robustness. To achieve these goals, they start by analyzing an intermediary protocol named synchronous G-GAP (SG-GAP). This protocol needs to detect that a node failed and to reinstate the transmission of it. To achieve this, it adds the necessary state information of the previous rounds. This allows an acknowledgment mechanism for the communication that ensures that the nodes can act upon this information. The next iteration of this protocol is the asynchronous G-GAP or just G-GAP. This protocol relies on an external entity to detect the failures. At the same time, includes less state information since it is not necessary. This protocol was evaluated by looking at the accuracy and how it improved the failure detection. They rely on a network simulator called SIMPSON and also in a network overlay simulator called GoCast. The accuracy of their results range from 80% to 95%, but they do not state the time it takes to converge. Therefore, we can agree that the proposals are interesting, but the results are not sufficient.

In general for the gossip-based approaches, it is fair to say that the approaches are more straightforward, effective but tend to be slower. The simplicity comes from their unstructured nature, where they do not rely on any structure to operate. This is a massive advantage for the implementation since it requires fewer assumptions and possibly fewer states to handle. On the other hand, it provides a small degree of uncertainty (e.g., mass loss) given the same unstructured nature. For the overall results, it is fair to assume that it is very robust. Looking at the different experiments, we can conclude that all the gossip-based approaches tend to have a good performance concerning accuracy. Moreover, the accuracy remains quite good even for dynamic and mobile networks. Regardless of this, the convergence time tends to be high in all results. This varies from approach to approach but given the core concept behind it seems that is prone to high convergence times. This is a necessary trade-off to keep in mind when choosing a gossip-based approach.

2.1.2 Hierarchical-based approaches

Hierarchical-based approaches commonly use tree structures in which the leaf nodes communicate the values with their parent node. These approaches are well known to the wired networks literature and have existed for a long time. They are known to be highly efficient to organize, aggregate and distribute data. The hierarchical-based approaches rely on a topology to work. This topology can be previously defined or defined at runtime. For wired scenarios is fair to assume that previously defined topologies are reasonably straightforward, assuming that the information of all the nodes is present and

that the nodes in the network will not vary with time. For more dynamic scenarios, including wired and wireless, the topology could be defined at runtime. This definition could be supported by another parameter as location, node capacity, antenna range or any other parameter to assist and ensure the efficiency of the topology. We can assume that for wireless scenarios could become more complicated given mobility. This would force the topology to re-arrange and re-structure itself constantly or to present a mechanism to minimize this process. Therefore, it is fair to assume that how the topology is managed has a significant impact on the approach. To apply a hierarchical-approach over a network, it is needed to rely on the topology mentioned above. This topology is usually built creating a group of nodes named clusters. Each cluster has a cluster head or cluster leader, which can be predefined or elected at runtime. Once the topology is in place, the exchange of information either aggregation or distribution is done between parent and child nodes. This is done recursively until it reaches the root node of the hierarchy. These approaches consider a mechanism of either pulling data or pushing data with their nearby nodes. Usually, the pulling data mechanism is done for aggregation. The advantages of hierarchical based monitoring algorithms are that they provide a fast convergence of the monitored property and produce less traffic. The disadvantages of these solutions are that they are prone to errors in the event of a crash in the network. This means that it does not perform efficiently in a highly dynamic environment.

We can mention the work of Stingl et al. (Stingl, Gross, Nobach, Steinmetz, & Hausheer, 2013) named BlockTree, which proposes a fully decentralized location-aware monitoring mechanism for MANETs. The idea is to divide the network into proximity-based clusters, which are arranged hierarchically. Therefore, the information flow is not centered around the pair parent and child, but it is centered around the relations between dedicated physical areas. Each cluster or block aggregates the data respecting a property. For this, each node knows and maintains information about their physical coordinates. The proposal analyzes three different components to achieve their goal, these are: a) the routing procedures b) the logical network partitioning c) and the approach itself, which is separated into a concentrating approach and a planar approach. The routing procedures are similar to geographic routing. To ensure the efficiency of the routing procedures in the given scenario, they get inspired specifically in receiver-based contention schemes (Füßler, Widmer, Käsemann, Mauve, & Hartenstein, 2003). They considered the routing procedures of receiver-based position unicast and receiver-based area dissemination. These procedures rely on the concept that each sending node broadcast the message without taking any resolution about the receiver. Then each node that receives a message calculates the suitability compared to other nodes to forward the message. This is achieved by interchanging specific information to allow this behavior, like the position of nodes, an identifier, a timeout and other relevant information. The logical network partitioning is achieved by creating blocks. The blocks are defined by the radius of the range of the nodes. This allows that all the nodes within a block can communicate with each other within one hop, and communication among different hops takes more than one hop. The concentrating approach, referred to as C-BlockTree, consists of two phases. The first phase is to collect data towards the root node and the second phase is to distribute the data relying on the hierarchy. The planar approach, referred to as P-BlockTree, consists of a simplified version of C-BlockTree, trying to maximize the communication and creating a more flexible approach. The difference with the previous approach is that the broadcast of data is not done to a single node but a group of nodes. Finally, the proposal was evaluated with the simulator “PeerfactSim.KOM”. The first round of test, they try varying the network size but keeping constant the number of nodes, for this, the accuracy decreases

as the network size increases due to the sparsely populated areas. Meanwhile, in a second test with dense networks, the accuracy increases as the network size increases. These are excellent results, but due to the need for the nodes geolocation, this has a high impact on the nodes network consumption.

We can also cite the work of Battat et al. (Battat & Kheddouci, 2011), which proposes a hierarchical monitoring for ad-hoc networks named HMAN. They propose a hierarchical approach based on the routing protocol OLSR to collect monitoring data over the network which is intended to manage the network itself. The approach consists of three main points. The first point is the construction of a hierarchical organization, which relies on a weighted approach for every node. Each node calculates its weight based on the node-degree, the cumulative monitoring time, the energy consumption, the total average distance between the node as a monitor and its members and the storage capacity. Based on this calculation and preset thresholds, the node has a specific type. If the weight is greater or equal than the preset threshold, then it is a monitor level 1. If the weight is greater or equal than the preset threshold and it has more or equal to two neighbors who are monitors level 1, then it is a monitor level 2. If the weight does not meet the previous criteria, then it is an agent. The second point is the monitoring approach which relies on three stages. The first stage is the data collection, and this is achieved by following the previously defined structure. Therefore, the agents send the information randomly to a monitor level 1 or level 2 in reach. The second stage is the analysis of the collected data. This is achieved by the monitoring level 1, which filters, analyzes and builds a monitoring report. This report is sent to the monitors level 2. The third stage is the data storage. This is achieved by relying on the monitors level 2. Then, if an agent requests a report, it is the monitors level 2 which find the requested information among them. Finally, the third point is the maintenance where they defined specific behavior for the nodes to react based on particular situations. Later they evaluated their approach relying on the simulator NS-2. Their results show that their approach does not affect the network performance and it does not add overhead due to their small number of generated messages. Although this, they do not analyze the accuracy nor the convergence time of the approach.

Another exciting work to mention is the one from Krishnamurthy et al. (Krishnamurthy et al., 2010), which analyzes a simple hierarchical protocol for performing a network size estimation. They propose the analysis of the protocol named GAP (stands for Generic Aggregation Protocol) (Dam & Stadler, 2005). This protocol relies on a cited work from Dolev et al. (Dolev, Israeli, & Moran, 1993), which is a self-stabilizing Breadth-first search (BFS) construction algorithm. From this work, they make an extension and an asynchronous adaptation. The protocol is self-stabilizing in the sense that a BFS tree is formed stable, the aggregated data is static, and the root node has the correct data. One of the assumptions from the work of Dolev is that the root node never fails and executes a different program than other nodes. Each node has three associated registers, which are the level, the partial aggregate and the believed parent node in the tree. In the case of the GAP protocol, their improvements are manifold. They start by relying on message passing instead of shared registers. Then each node maintains information in a data structure about its child nodes in the BFS tree, and this is done to aggregate data effectively. Finally, the protocol is event-driven, meaning that only upon request triggers the necessary mechanism to aggregate, this allows it to reduce the traffic overhead. The GAP protocol relies on an execution model, which allows it to asynchronously pass messages for tasks like new neighbor, failure detection, update an entry, updating weight and timeout. The overall execution of the GAP protocol relies on a table of siblings and neighbors, which can operate based on different

policies. These policies are conservative, cache-like and greedy. Based on all these definitions of the GAP protocol, an assessment of its performance is made. The validation through simulation was done using a discrete-event simulator. The simulations performed runs for different combinations of number of nodes, average node degree and number of runs. The results show that the accuracy of the protocol decreases with an increasing network size, but it increases with an increasing degree of the network and number of runs. The accuracy ranges from 50% in the worst case scenarios (big network size) to 98% in the best case scenarios (big network size but high network degree and runs). Although the accuracy results seem promising, the study did not take into account the convergence time.

In general for the hierarchical-based approaches, it is fair to say that the approaches are complex, effective but tend to be fragile although theoretically faster. The complexity is derived due to the need to follow a structure. This adds layers of complexity as having a layer to keep the hierarchy functioning, another layer for aggregating and fallback mechanisms in the case of primary nodes failing. This does make it more complicated, but at the same time reduces the uncertainty by defining the proper mechanisms for the hierarchy to work. The results, in general, seem promising. Theoretically, the approaches should be faster and should be able to converge with outstanding efficiency. Sadly, none of the cited works analyzed the convergence time, although most of them mentioned the theoretical efficiency regarding the time of hierarchical structures. Aside from this, the accuracy results seem to be very promising. It is fair to assume that in low dense networks the approaches have difficulties to work. However, on the other side, it seems that for a balanced ratio between the number of nodes and network size the results are excellent.

2.1.3 Hybrid-based approaches

Up to this point, we can agree that the previously mentioned approaches have essential trade-offs that they need to work on. On the one hand, we have the gossip-based approaches which are highly efficient in dynamic and mobile scenarios. The accuracy is relatively high in most of the scenarios that the cited works realized experiments on, but the convergence time is also relatively high in all cited works. On the other hand, we have the hierarchical-based approaches which are highly efficient to aggregate data based on a pre-defined topology. Although their efficiency, they are fragile due to the same topology in dynamic and mobile environments. There are attempts to improve this, but these approaches incur in less energy efficiency or adding more components that make the approach unnecessary more complex. Theoretically, the hierarchical approaches are very fast to converge and achieve high accuracy. Therefore we could assume, based on these two approaches, that a middle point could be of high interest. For this, we are assuming that we want to get the efficiency in convergence time and the robustness to withstand dynamic and mobile environments. On top of this, this middle point should have outstanding accuracy. Based on this assumptions and requirements, we could try to derive an approach which achieves this. Therefore, this categorizations tries to encapsulate this. The hybrid-based approaches could be either a gossip mechanism supported by a hierarchical approach or either a hierarchical approach supported by a gossip mechanism.

Therefore in this categorization, we can mention the work of Subramaniyan et al. (Subramaniyan, Raman, George, & Radlinski, 2006), which proposes a reliable and scalable gossip-enabled monitoring

service named GEMS. This approach provides a monitoring service that addresses the problematics of clustering, failure detection and performance monitoring with low overhead due to the gossip-based approach. The service relies on a distributed consensus mechanisms to detect failures with fast reaction. It also provides an API (Application Programming Interface) to allow users to connect with the service and share information which allows scalability and ease of integration. The proposed approach relies on basic information derived from the gossiping procedures. The data structures maintained for each node are the gossip list, suspect vector, suspect matrix and live list. The gossip list contains the time between the current node and the other nodes. The suspect list contains a list of booleans to determine if a node has failed or not for each node. The suspect matrix is the combination of all the suspect lists in the system. Finally, the live list contains the health status of all the other nodes in the system. The hybrid approach of this work is the proposal of a layered gossiping design. This approach was inspired by the works of Sistla et al. (Sistla, George, Todd, & Tilak, 2000; Sistla, George, & Todd, 2003), which focuses on various flat and hierarchical gossip-based failure detection approaches for distributed systems. The layered gossiping is formed by groups of as much as three nodes per group. For example, this means that they form a hierarchy of N layers or levels, for a network with 3^N nodes. Therefore for a network with 27 nodes, it creates a hierarchy of 3 levels. It is worth mentioning that due to this hierarchy, the upper layer also contains a maximum of 3 nodes. Therefore there is never a root node for this hierarchy. This approach relied on a consensus algorithm to determine when a node has failed, by propagating the live list of a group to all the nodes. It also relies on a timeout and higher groups consensus to detect network fragmentation and group failures. Although they state this approach is valid, it is not clear enough on how the consensus works. From this base, they dedicated a great effort in defining the API to provide a flexible and well-defined framework and service. Later they evaluated the efficiency using 150 physical nodes. The average results for the consensus mechanism to converge are around 100ms. They also reviewed another scenario where the hierarchy is assembled, but new nodes are added, in this case as the number of nodes increased, the new node insertion became slower but not by much and is still in the magnitude of approximate 800ms. The convergence time is very impressive, although they do not analyze the accuracy. It is worth mentioning that this work is focused on clusters in wired networks.

As the writing of this thesis, there are no known hybrid approaches focused on wireless networks. Based on this limitation is fair to assume that more work is needed in this area. Although promising results are coming from the wired networks. They seem to work faster but the accuracy still a remaining open question in the context of wireless networks and more specific for MANETs.

2.2 Distributed Monitoring for MANETs

The distributed approaches deal with networks that do not have a central coordinator just like decentralized approaches. The difference is that distributed approaches have the procedures, functioning, and storage distributed across all participants in the network. This definition sometimes gets somehow confusing due to the same distributed architecture. Therefore for our purposes, we say that a network is distributed if all the participants can, have the opportunity and the decision to participate in the procedures, functioning, and storage of the activities of the network. This means for example, in the

context of monitoring, there might be the need for aggregating a value. Therefore an aggregation might be done by only one node, but if all nodes have an equal probability to aggregate this value, therefore is distributed. Based on this description, we try to look at some works that are alongside this line.

We can start by mentioning the work of Belfkih et al. (Belfkih, Duvallet, Sadeg, & Amanton, 2017), this approach is applied to wireless sensor networks (WSN), but they share the same resource limitations as MANETs. They propose an improvement over classical SQL-Like query language processors for WSN to query values of the network. The architecture proposed consists of three types of nodes: a) base station b) relay node c) and child node. The base station consists in a cache memory for measurements, an arrival query list, the deadline controller, the transaction scheduler and the data validity controller. The deadline controller makes sure that every response to the query is made within some defined time parameters. The relay node consists of everything the base station consists except the cache memory. Finally, the child node consists of the deadline controller and query processor. On top of this architecture, they propose a query processing plan, which is achieved with three procedures: a) topology creation and relay nodes selection b) query dissemination c) and query response collection. The relay node selection is made by the nodes themselves, and it is done by an auto-selection process proposed in the work of the same authors in (Belfkih, Duvallet, Amanton, & Sadeg, 2015). They relied on the RPL routing protocol proposed by Winter et al. (Winter et al., 2012). Through this process, a tree topology called DODAG (stands for Destination Oriented Directed Acyclic Graph) is built. The query dissemination procedure is in charge of querying the required values if the query has not done in the past, if it does it fetches the already stored value. If a more recent measurement is needed, the query is placed in the active list and queried again. The query process is handled by the deadline controller and the transaction controller. The query response collection procedure is done through the same tree topology. For this procedure, the data validity controller is used. They evaluate their proposal relying on some relay nodes that use some scheduling algorithms to process the responses. It is important to mention that their monitoring results had a convergence time between 10s to 20s. Although is a quite interesting approach, it is not clear if they define their proposal as decentralized or distributed.

An interesting approach is the work of Battat et al. (Battat, Makhoul, Kheddouci, Medjahed, & Aitouazzoug, 2017). They define a trust computation method for a node to determine if neighboring nodes are showing malicious or selfish behaviors. This is done by proposing an architecture that is logically divided into clusters. Then each cluster elects a monitor head. The election is based on a multi-criteria method defined as the interchange of trust information about other nodes within the cluster. Then the node, which has the higher trust level is elected as the head of the cluster. They argue that his multi-criteria method is more efficient and fair over other approaches categorized as unique criterion based election approaches, where the same node can frequently be elected. The criteria used is composed of the trust value coefficient, the processing power, the energy level and the storage capacity. The trust computation method is based on each node observing the trace behaviors of its neighbors. Based on these interactions, the nodes can update a trust value for each node within range. The initial trust value of every node is equal to 0.5 where this value is within the range of [0, 1]. They define multiple rules that mandate how the nodes update the trust value. A node is considered malicious based on their trust value from the following conditions:

a) Falsifies monitoring policies

b) Unnecessary traffic generation

c) Broadcast non-existing nodes

d) Send fake data, which other can check based on their data

e) Broadcast false alarms

f) Does not contribute in all monitoring tasks (like data forwarding and data storage)

In their simulations, they provide a low message overhead, but the election methods could be weak compared to the leader election of a consensus algorithm. Their results are focused on the number of messages sent, number of malicious and excluded nodes but there is no analysis for the convergence time or accuracy of the approach. Nonetheless, they provide an interesting approach based on trust for the monitoring of a MANET.

One of our proposals explained in depth in Chapter 4 is inspired by the blockchain technology. The blockchain technology is explained in details in Section 4.1.3, but this also represents a distributed system. The most famous application of the blockchain technology is cryptocurrencies, given that the concept itself was derived by the proposal of the first cryptocurrency. Regardless of this, there are some attempts to introduce this topic in different contexts of computer science. To our knowledge, no studies are referring to a blockchain-based approach focused on the monitoring of networks. There is an interesting approach worth mentioning that is the work of Selimi et al. (Selimi, Kabbinale, Ali, Navarro, & Sathiaseelan, 2018). It is fair to say that this work also assumes, to the best of their knowledge, that there are no blockchain deployments made in wireless mesh networks. This work focuses on the Linux Foundation blockchain, which is called the Hyperledger Fabric project (HLF). The idea of this work is to deploy the HLF into a production wireless mesh network. They focus on a community mesh network (CMN), which are a particular case of the wireless mesh networks. They rely on an existing CMN called Guifi.net, which is a community-driven effort to create an ISP-less Internet with 34,000 current participants around the world. The HLF is not designed for the moment for wireless devices, and it is designed to work as a permissioned blockchain, which means a private blockchain. A private blockchain is the equivalent of a private network of any company. Therefore the primary aim of the study is to show the feasibility of the deployment and to measure the resource consumption of it. The HLF relies on the following components: a) peers (endorsers or committers) b) ordering service (in charge of the consensus) c) chaincode (application logic) d) and channel (confidentiality abstraction). They evaluate their proposal by measuring the transaction latency and the resource consumption of the different types of nodes of the network. The transaction latency is out of the reach of this thesis, therefore is not taken into account. The resource consumption is interesting, and it shows that for some nodes the CPU consumption during the experiment can range from 70% to 96% during a 60 seconds time window. It is fair to mention that HLF relies on a modular consensus algorithm, i.e., it is up to the manager to decide which one to use. Therefore, it is not mentioned on which consensus algorithm they relied on, which could have a direct impact on the CPU consumption. This being said, although the results are impressive, they could be discouraging for a limited resource network like a MANET. For this is essential to understand that although the actual implementations are resource intensive, it does not means that the blockchain technology itself has to be resource intensive. We

firmly believe that the blockchain technology is a distributed system in all possible aspects which has incredible potential for many applications. Although to work effectively for specific applications, it is necessary to comprehend and choose the corresponding components based on the limitations.

For the distributed approaches, the proposals are more distributed into specific scenarios since it may require a given complexity to formulate them. Nonetheless, different works have shown the importance and the efficiency of them. We firmly believe that given the context of distributed monitoring and the blockchain technologies as a solution for distributed systems, there is a possible match for them to work. It is interesting to note that other approaches have mentioned and suggested the use of a consensus algorithm to agree on the correct data in the monitoring systems. The cited proposals were evaluated as having different goals in mind, and therefore it is hard to compare them with the decentralized approaches. We can mention that the convergence time for one of the proposals is more acceptable than the results for the gossip-based approaches. Nonetheless, there is room to do more experiments focused on the convergence time and accuracy.

2.3 Remaining Open Challenges

As mentioned in the previous sections, there are multiple open questions and different aspects to consider to provide robust and efficient solutions for decentralized and distributed monitoring. Looking at the state of the start it seems that decentralized solutions have existed for more time and therefore they are studied more in depth and with more aperture. Distributed solutions have become a big topic of discussion in recent years due to the flexibility they could potentially provide by removing coordinators even in decentralized approaches. Even though, the proposed solution should fit the scenario where it is being applied. We believe that is worth looking at novel technologies as blockchain to see if they can be applied and if not, maybe an inspired mechanism can derive an excellent proposal and solution. Therefore the remaining open challenges and main contributions to highlight in this thesis are the following:

- In the decentralized literature, there are great proposals for gossip-based approaches and hierarchical-based approaches. However, it seems that space is narrow for hybrid-based approaches. Based on the results of both categories, it is fair to assume that a more efficient proposal could be derived when appropriately combined. The proposal needs to rely on a hierarchy that allows it to aggregate results efficiently. Nonetheless, it needs to strongly rely on gossip mechanisms to disseminate data and to support the hierarchy when its frangibility is affected by the mobility and the dynamic properties of the MANETs. This work is presented in Chapter 3, by the proposal of a hybrid-approach for monitoring a MANET.

- In the decentralized and distributed literature, there are reasonable proposals, but it seems that their mechanisms are not fully explained or defined. Many of the proposals make a great effort in illustrating and providing examples, which help the reader to understand the problematics and the proposals. It can be observed that the proposals have detailed approaches, but they usually lack formal and robust mathematical definitions. This is desired to design on top of a

more robust mathematical bedrock, which inherently will provide a more robust proposal. A set of mathematical definitions is desired to present in a deterministic and unambiguous way the proposed solutions. This work is shown with the support of automata in Chapter 3, but it is expanded and detailed in more depth in Chapter 4 by presenting the definitions to express our monitoring properties in a distributed way.

- In the distributed literature, the proposals were evaluated using different assumptions, techniques, and simulators. Although, many of the proposals seem to not look into convergence time nor the accuracy of the approach. This could be done intentionally due to a given reason, but none of the works explicitly mentioned their motivation not to test this. We believe is important, and based on other works and even the works in the decentralized literature, these measurements should be the core for performance evaluation of any proposal. The performance evaluation can be divided into validity and timeliness. The validity refers to the accuracy of the approach and the timeliness of the convergence time or how much time the proposal takes to provide a global view of the network. This work is exposed in Chapter 4 by the proper evaluation of the performance of our proposal.

- The context of distributed systems, which encapsulates the distributed monitoring context, has gained recent attention due to the blockchain technology. This novel technology has some different observations from the outsiders and academia ranging from critics to praises. Some of these observations, sometimes are taken out of context and therefore creates some debate around the technology. We do not intend to analyze nor comment on its application for the cryptocurrencies. However, we think that this technology, defined and implemented in the proper context, relying on predefined assumptions and limitations which dictate the implementation and the final results could provide exciting results. For this is essential to mention that due to the cryptocurrencies blockchain implementations, the blockchain technology has the unfortunate reputation to be resource intensive. This is somehow true, but it also depends on the complexity of the current implementations. We believe that a slimmer and dedicated implementation could overcome this and in overall provide efficient and robust results. This work can be seen in Chapter 4 by the proposal of a distributed monitoring approach inspired by blockchain.

- From simulation literature, it has been said that many works in the network literature do not make the experiments repeatable. Repeatability is a core concept of science, and they claim that if an experiment is not repeatable the credibility of it is affected. We strongly support this claim and also, we could see the same behavior it has been described in the works of state of the art. Although sometimes they explicitly define which simulator or emulator was in use and the parameters for the emulations, they are not always clear. Sometimes, they simulate the experiments but only partially define the parameters of the simulation. More important, the amount of scientific work which provides the code for further tests from scientific colleagues is nearly inexistent. This is claimed by multiple simulation surveys, and we could conclude the same. For this, it is needed to expose more the implementations alongside the simulators or emulators and the parameters used. This work can be seen along all the thesis, in Section 3.3 and Section 4.3 we outline all the parameters used for the emulations. In Chapter 5, we detail the emulator used for all the experiments, which is publicly available and already in use by other researchers, to improve the overall credibility of our work.

3

Decentralized Monitoring

Contents

Though MANET monitoring is crucial, it faces several issues mainly due to the inherent nature of such networks, i.e., the nodes mobility and the resources constraints. Some approaches have been proposed with a centralized node, named a coordinator, in charge of the monitoring. Unfortunately, such approaches suffer from many drawbacks due to energy efficiency, Internet access, infrastructure or other parameters, which render such approaches not always applicable.

Monitoring a network implies to obtain a global view of the system through attributes to be observed. In the literature, within a centralized approach, a central node is defined to collect and disseminate the observations. In (Cormode, 2013b), the authors review the different communication mechanisms to optimize the data interchange. Such approaches are effective for certain types of topology, while they are not at all in the presence of dynamic topology. To overcome this limitation, decentralized approaches have gained considerable research interest.

To monitor a network, it is often needed to be able to find out the status of the network or some of its properties. Generally, an intuitive approach is to define a central node as a coordinator for storage and processing of the observations. This is notably proposed by (Cormode, 2013b), where the author surveys the different communication mechanisms for an optimal data interchange. These centralized architectures are effective for certain types of topologies but become critical when considering dynamic topologies. This is why there has been a lot of efforts on decentralized monitoring.

The benchmarks provided by (Stingl et al., 2012) established some non-functional requirements of a decentralized monitoring mechanism. Performance, costs, fairness, scalability, robustness, and stability are the quality aspects to be evaluated. Performance relates to the accuracy of the delivered results and how fast they are delivered. The costs refer to the overhead by the communication or processing of the data. Fairness can be analyzed in respect of performance and cost. The scalability refers to the ability to work in large and dense networks as well as in small networks. The robustness deals with the behavior of external and unpredictable events. And the stability is the ability to address the random behavior of autonomous nodes. Based on these requirements, existing solutions rely on gossip-based or hierarchical-based approaches. Each of them provides a specific set of features and also downsides for the monitoring process, as studied by (Stingl et al., 2012). This work studies in a rigorous way the features and downsides of these solutions. Gossip-based approaches demonstrate their robustness and stability in dynamic scenarios and changing topology. Nonetheless, depending on the scalability, the cost and performance can be impacted. On the other side, hierarchical approaches show an efficient performance, cost, and scalability, although the robustness and stability may decrease in dynamic scenarios. This shows that the two major categories perform very good under different characteristics, requirements, and constraints of a network. Therefore, we are convinced that a new algorithm could be derived from these two approaches for wider scenarios. Our contributions in decentralized approach are manifold.

- We propose an enhanced algorithm denoted as hybrid, as it is a combination of gossip and tree based approaches for decentralized monitoring of mobile ad-hoc networks in dynamic context. We define an architecture combining gossip-based and hierarchical-based algorithms for query dissemination and data aggregation. The gossip-based approach allows to disseminate the query and in the process to build a virtual hierarchical topology (VHT) for a time window. Once the query is disseminated through all the network, with the support of the VHT, a hierarchical-based aggregation takes place. To provide robustness and stability, we provide intermediary gossip-based mechanisms to complete the aggregation even in dynamic environments.

- The introduction of a multi-root node approach enhancing the monitoring process and minimizing network fragmentation. By defining several root nodes, we considerably improve the accuracy

results. This enhancement also allow us to explore the possibility to consider a distributed approach to achieve the same result.

- We have conducted a large campaigns of experimentation to assess the results of our proposal. To address the scalability issue, we ran a large range of experimentation which allows to be more confident in the obtained results. We decided to do three campaigns of experiments, the first was designed to test our approach, the second was designed to vary as much emulation variables as possible to ensure the efficiency of our approach, and the final and third was designed to enhance the approach by increasing the number of root nodes.

Based on these contributions, we intend to provide a solid starting point for more hybrid proposals. In the literature of MANETs, there are a lot of areas of interest which are being highly studied. Among this interest, we can find the monitoring services, which as we mentioned, have gained a lot of attention in the decentralized algorithms. We hope for more researchers proposing new ideas and algorithms for gossip-based, hierarchical-based and hybrid approaches. We truly believe that our monitoring protocol along the hybrid algorithm will work as a solid ground for further ideas.

This chapter is divided as follows: In Section 3.1, we give the basic concepts and notions about gossip-based and hierarchical-based monitoring approaches. Then, in Section 3.2, we detail in depth the hybrid monitoring architecture, defining the query and aggregate procedure, and the respective enhancements to make it more strong. Finally, in Section 3.3, we describe our large campaign of experiments where we highlight the importance and effectiveness of our approach.

In this chapter, we expose the work which was presented in the following publications: (Alvarez et al., 2016, 2017a, 2017b).

3.1 Preliminaries

Network monitoring is an extensive field of interest. Cormode describes it as “a number of observers making observations and wish to work together to compute a function of the combination of all their observations” (Cormode, 2013b). For our purposes, the observers are the nodes in the MANET. The goal is that all network nodes compute a value $f(t)$ in a given instant of time t in a collaborative way. The function $t \mapsto f(t)$ [$\mathbb{R}^{+*} \rightarrow X$, X being the domain targeted by f], for our purposes, is a linear and non-complex function like the average CPU, average RAM usage, or any other nominal value. Given these definitions, we can imply that every node in the network is a point of observation of a set of properties. These observations need to be analyzed in a network scale to be able to determine the health and state of the overall network for a given property. It is important to mention that the monitoring process can be triggered by an event, for example a new value of a monitored property or a query to compute the global view of a property. Also, but not limited to, it can be triggered by a periodic polling. We present the basic terms for the reader to understand the following sections in this chapter.

3.1.1 Types of Monitoring

The classification of the monitoring process has been studied in (Battat et al., 2014). For the purposes of this chapter, we consider the two major types: centralized and decentralized. The centralized type of monitoring, as stated by (Cormode, 2013b), is when all the nodes report their observations to a central entity, named centralizer or coordinator. This entity processes all the observations from all nodes to reach a global view of a property of the network. This approach is the most common approach specially in static networks. The decentralized approach, the opposite of the previously defined approach, deals with networks where there is no centralizer entity. This implies that the network by itself needs to achieve a global view of a property of the network. The preferred approach is to get the global view of a property of the network, and then disseminate it through the entire network to assure that is available to all the network. As stated by (Stingl et al., 2012), the more noticeable approaches are currently gossip and hierarchical.

Gossip-based approaches

The gossip approaches are based on the gossip or epidemic algorithms. These algorithms define that each node in the network contains a set of reachable nodes. Therefore, these algorithms rely on selecting a subset of reachable nodes to forward the data packet. The subset of reachable nodes can be chosen randomly or based on other criteria. It can also be one or more nodes. There are multiple types of gossip techniques, a more detailed study can be found in (Chakchouk, 2015). In this publication, the focus is on routing protocols. But it explains how gossip based protocols implement their goals through different techniques and properties. For example, energy efficient gossip based protocols inter change information about their batteries and energy source, so the overall algorithm can be more efficient. Probabilistic gossip based protocols use the probabilities of meeting a node to their advantage. A general assumption in this kind of protocols is that the nodes requires to be synchronized. Epidemic algorithms try to forward the packet not only to one but to multiple nodes. As the name implies, the objective is to disseminate information by infecting as much nodes as possible to ensure an efficient dissemination. These algorithms are also considered a subcategory of the gossip algorithms. Flooding is the most common and simple algorithm for epidemic algorithms. Gossip-based monitoring algorithms have the advantage of being highly stable and perform better in increasing dynamic networks. For proactive approaches, the network overhead can be higher but it has faster results. For reactive approaches, the results are slower and could potentially chose inefficient routes but the network overhead and overall resource utilization is lower. In general, both approaches may generate lots of traffic and, under certain scenarios, require more time to compute a value.

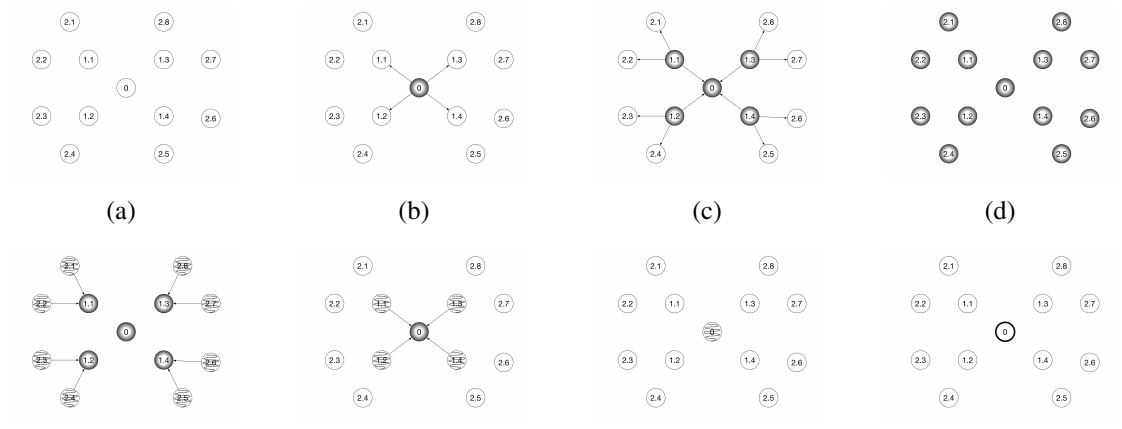
Hierarchical-based approaches

Hierarchical approaches commonly use tree structures in which the leaf nodes communicate the values with their parent node. The exchange of information among the nodes usually is done recursively from the root node towards all the leaf nodes or from the leaf nodes towards the root node. These

approaches consider a mechanism of either pulling data or pushing data with their nearby nodes. The pull data mechanism refers to the process of a parent node requesting information from their child nodes. Meanwhile, the push mechanism refers to the process of a parent node sending information to their child nodes. It is important to mention that both mechanisms can be used in parallel to improve the overall communication exchanges. To apply a hierarchical algorithm over a network, it is needed to build the topology before being able to monitor the network. The topologies can be built using different procedures. The most basic procedures are predefined topologies. It can be easily noticed that this procedure might sound easy to define for static and small networks. But in numerous and dynamic networks this approach is not feasible. Another approach is to define clusters based on location or based on the nodes capabilities. Then each cluster has a cluster head, and then based on this structure a hierarchical topology is defined. The advantages of hierarchical based monitoring algorithms are that they provide a fast convergence of the monitored property and produce less traffic. This is due to the fact of the defined structure, which allows the overall procedure to work more efficiently. The disadvantages of these solutions are that they are prone to errors in the event of a crash in the topology. This means that it does not perform efficiently in a highly dynamic environment. In order to work efficiently in a dynamic network, the algorithms require extra mechanisms to re-configure the topology. These mechanisms can have a negative impact on the overall efficiency of the topologies.

3.2 Treesip: A Hybrid-based Monitoring Approach

The hybrid algorithm architecture herein proposed, defines an architecture based on a temporary tree structure supported by two mechanisms: the query and aggregate procedures. A root node is randomly selected and triggers the monitoring process. Given the algorithm, we rely on an event-driven trigger, which will be a manual query performed by an outsider entity, either a network operator or a node in the network. This node will start by propagating a query and assembling a temporary tree or hierarchical topology. The idea is to combine a gossip approach and a hierarchical approach to achieve the monitoring of a property of the network. The communication between the nodes to achieve the monitoring of the network is achieved through a package previously defined. The objective is that a root node starts the monitoring process by propagating a monitoring query in a gossip approach. The approach chosen is described as epidemic, given that the idea is to share information in an efficient, fast and simple way. Each hop, the nodes exchange information creating a hierarchical topology valid only during this monitoring process. Then based on this topology, the nodes start aggregating the information by sending their results to the parent node. If the parent node is out of reach due to mobility or offline, the approach then triggers a gossip approach to deliver the message and continue the aggregation. Once the aggregation is done and has reached the start node, there is a global view of the measured property and the hierarchical topology is no longer usable. If the process starts again, a new hierarchical topology is derived. The purpose is to establish a virtual hierarchy during a time window only, the duration of the monitoring process, to ease the analysis of it and the global view of the property.



(a) (b) (c) (d)

(e) (f) (g) (h)

Figure 3.1 – Example of the hybrid monitoring approach

3.2.1 Treesip in a nutshell: a detailed example

The overall process is exemplified in Figure 3.1. In Figure 3.1a, the network is in the initial state. Only for the purposes of this example, we are assuming that the nodes in the network are not moving and that they are synchronized for simplicity purposes. Node 0 starts the monitoring process and changes its state to query state and it disseminates the query to neighbors within its range, as it can be seen in Figure 3.1b. The query requests the computation of the value $f(x)$ and contains network information for the hierarchical topology. In this case, the query is sent to node 1.1, 1.2, 1.3 and 1.4. Then each of these nodes repeats the same process, changes its state to query state and disseminates the query. If we focus on one particular node, e.g., 1.1, we can see that the query is communicated to 2.1 and 2.2, as it can be seen in Figure 3.1c. After that, all the network converges into the query state, as observed in Figure 3.1d. Since the edge nodes are reached, these nodes change from query state to aggregate state and send their aggregation results to the parent node. If we look at nodes 2.1 and 2.2 in Figure 3.1e, both share node 1.1 as the parent node, so each one of them sends their information to the parent node. Then in Figure 3.1f, we can see that 2.1 and 2.2 are removed from the hierarchical topology, and that 1.1 changes his state to aggregate state. In this moment, node 1.1 aggregates the results from 2.1, 2.2 and itself, and sends it to his parent node, i.e., node 0. Then the process repeats itself with node 0 in Figure 3.1g. Node 0 receives the aggregation results from the nodes and aggregates the results from node 1.1, 1.2, 1.3, 1.4 and itself. Finally in Figure 3.1h, the network converged the monitoring process and node 0 has the global view of the monitored property. The derived hierarchical topology that was built to calculate the queried value, is dissolved and no longer valid.

3.2.2 Network protocol internals: topology and procedures

Virtual Hierarchical Topology

We define the process to build a hierarchical topology for a time window to support a hierarchical based aggregation of the monitored values through the network. Given the fact that it only exists for a

limited window of time, we can say that it is a virtual entity. Due to this, we refer to it as the virtual hierarchical topology (VHT). The VHT concept has been introduced in (Huang, 2005) and adapted by Google in one of his patents for cache nodes (Google, 2014). The advantage of the VHT is its simple packet forwarding configuration. Each child node only forwards data packets to its parent node. The message propagates in such a manner until it reaches the root node. During the monitoring process, nodes exchange several packets. As it is later detailed, some of them send queries, some others receive them. The senders are called *parent nodes*, the receivers named *child nodes* of their *parent*. When the (parent) nodes send such packets, they are disseminated throughout all the network until reaching the edge nodes of the network. By that process, topology information are exchanged between the parent/child nodes for a specific time window. The VHT is then built. The following steps summarize the VHT construction:

i Each source node (chosen by the experts to aggregate the collected monitored values) sends a query to its neighbors. A timestamp information labels the time window.

ii Each node (that is not an edge node) receiving a query forwards it if not already received before. The hierarchy parent/child and the timestamp are stored.

iii An edge node receiving a query does not forward it.

This concept is very important since we here rely on a hierarchical approach. Therefore, we need a topology built before we are able to aggregate our results. But our goal is to utilize this hierarchical topology only for a limited and minimal amount of time so we minimize its frangibility. For that, we use the VHT only during a limited time during which the monitoring is done. It is a virtual representation since, given the mobility, it will probably not be able to keep the same topology physically. The VHT validity time starts with the monitoring process and ends with the convergence of the process, both instances of time considered from the root node. This way, we can aggregate efficiently and for every new monitoring process, a new VHT is derived.

Query Procedure

The query procedure refers to the process of propagating in an epidemic way the monitoring packet. This state goal is to disseminate the query and the VHT layout to allow the nodes in the network to do an accurate and efficient aggregation in the next state. This query is forwarded in an epidemic approach to the nodes in the relay set. Let us call $sender$ to the node sending the query and $receiver_i$ to the receiving nodes of the query. In the exchange of information, $sender$ is the parent node and $receiver_i$ are the child nodes of $sender$, since they resemble the hierarchy of how the query is being propagated. This happens recursively until the edge of the network is reached. This process is what creates the VHT. It is a virtual representation, since given the mobility, it might not be able to keep the same topology physically. The packet is explained in depth in Section 3.3.1, containing the query itself but also the information to generate the VHT. This communicates all the network information to create the VHT, which is the foundation of the following state of the network. This process goes on until a node on the edge of the network is reached.

Along this state, there are some specific challenges to discuss.

(i) The first challenge is if a node receives more than one monitoring packet once it is already in a monitoring state. This implies that a node is in the range of one or more probable parent nodes. For this, the node always takes the first monitoring packet, and it discards all the subsequent monitoring packets.

(ii) The second challenge is, what if the propagation of the query is interrupted by a node that remains in a cyclic state. For this, we introduce a timeout for the packet to avoid these problems. This can be considered a time-to-live for the packet. The idea is to provide a mechanism to avoid loops in the communications. This means that a node, because of a race condition, cannot continue disseminating the monitoring packet, since all the nodes in his relay set are child nodes of another node in the VHT. For this problem, the timeout is triggered and once reached, this node starts the aggregation process by sending its result to the parent node. This way avoiding infinite loops in the VHT.

(iii) The third challenge is the broadcast of the packet itself. Due to the nature of the simple epidemic dissemination approach, a packet is forwarded to the next hop of nodes but also to the parent node (because of the medium and geospatial temporality). To make good use of this, we decided that this works as an acknowledgment of the child node to the parent node. This way, the parent node has to receive n amount of acknowledgments. Based on this, it will know how many packets he should wait for before changing to the aggregate state. We are assuming that the sender and the receiver nodes of this query packet will be within range to make a successful exchange of packets. After this is achieved, these nodes could get out of range and this does not affect our algorithm.

It is worth mentioning that based on the work of Drabkin et. al. (Drabkin, Friedman, Kliot, & Segal, 2007), it is expected that the query takes from 2 to 5 hops to reach the majority of the network. This is calculated based on the coverage probability of a message m reaching all the nodes in the network. For our case in particular, it means that theoretically, the VHT will have from 2 to 5 levels.

Aggregate Procedure

Once the data is disseminated up to the edge of the network, the edge nodes change from the query procedure to the aggregate procedure and start sending recursively their information to their parent up to the root node. This process is an aggregation of all the data of a node and its children in order to collect the monitored values. The aggregation is computed in a hierarchical manner with a combination when required of a gossip fallback mechanism. A node computes, based on his own observations, the result of the function $f(x)$ received from the query procedure. This information is aggregated with the same information of the child nodes. In case of edge nodes, where this state starts, it is done only with information from themselves.

Along this state, there are specific challenges to discuss.

(i) The first one is when a parent node and a corresponding child node goes out of range from when they first met. When the child node sends an aggregate type of message and receives no acknowledgment, it triggers a route packet to the corresponding node. The time and hops necessary to take to find the corresponding node is highly dependent on the nodes mobility. For this, we rely on the gossip routing protocol of our approach. This tries to reach the parent node using a random gossip approach for routing the packet. This means that the forwarder of the packet selects a node randomly, from the available neighbors at the moment.

(ii) The second challenge is when a parent node is offline. It means that a child node already tried to route the package to him and still had no answer. For this, we propose that in the query state, a set of nodes are communicated to every child node for them to have an alternative path. Since the child node has the relay set of the parent node, he can fall back into one of these nodes to send the information. These other nodes will continue the aggregating process. Since it is a hierarchical approach, the relay set transmitted from a parent node is the set of parent and grandparent nodes of itself.

(iii) The third challenge is when a node receives a grandchild node aggregate information. For this, the node assumes that the child node is offline and that he is going to be aggregating that information. Given that the node does not know the information of how many grandchildren are going to send information, he also relies on the timeout before he sends his own aggregate information. For every grandchild packet he receives, he restarts the timeout to give time for additional packages. If the timeout is reached, he continues with his aggregation process. If more information is received, he forwards it to his parent and they are going to aggregate it ultimately by an upper node in the hierarchy. In the worst case scenario by the root node.

Gossip-based Routing Fallback Procedure

We formalize the gossip routing fallback process as the input/output finite state machine (FSM) depicted in Figure 3.2. This is an important part of the architecture since due to the mobility, it is expected to support the VHT. Every packet is uniquely identified by the concatenation of the unique identifier of the generating node (IP address) and the timestamp of the creation time (unix time). The set of states is $S = \{Initial, WR, DN\}$, WR refers to waiting for a message hello reply and DN refers to the end of the routing the message. We define the set of inputs as $I = \{Aggregate, Route, Hello, HelloReply\}$ and the set of outputs as $O = \{Hello, HelloReply, Route\}$. Then we define the transition as $Timeout$, which is any t that maximizes the delivery of the packet. The $Aggregate$ message triggers the routing process from the monitoring layer. The process aims at sending a $Hello$ message to its neighbors, the closest node replies with a $HelloReply$. Then it routes the packet to this node, theoretically the state DN should be the final state of the FSM. But there is a probability that, due to mobility, the packet needs to be rerouted again by it, therefore we leave the possibility open by the transition between DN and the initial state. If we received a $Route$ message, we first verify if it is addressed to itself. If this is the case, we route the packet internally to the monitoring layer, otherwise we continue the routing process.

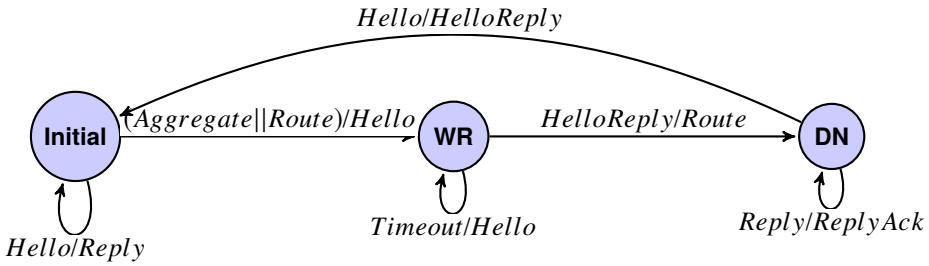


Figure 3.2 – IO FSM definition of the fallback gossip routing process for the aggregate procedure

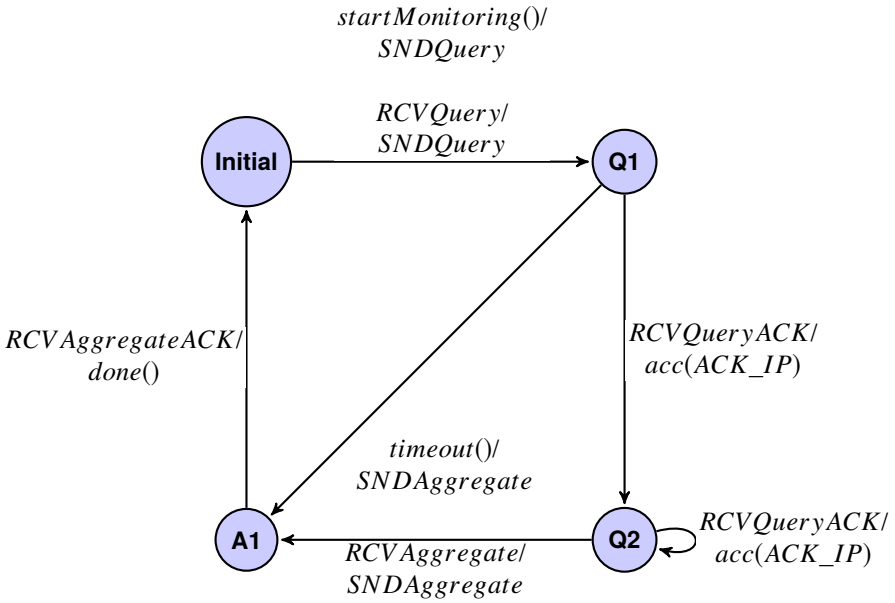


Figure 3.3 – Finite state machine of the protocol without mobility support

3.2.3 Network Protocol Automata-based Definitions

To support our architecture, we start by defining a protocol in a semi-formal way through a state automaton and also by defining the corresponding algorithm to ease the implementation and to provide mathematical background. The protocol definition, depicted in Figure 3.3, shows the expected behavior of the protocol to support as base ground for the hybrid monitoring architecture for a network without mobility. The set of states is $Q = (Initial, Q1, Q2, A1)$ where $Initial$ is the initial state. The states $Q1$ and $Q2$ refer to the query states of the network. And the state $A1$ refers to the aggregate state of the network. The internal operations of the automaton are startMonitoring(), acc(IP), timeout(), done() and error(). The startMonitoring() refers to the process of starting the monitoring. The goal of this process is specifically to inform the nodes what function(s) (fn) has to be monitored. The acc(IP) refers to the process of the node of accumulating the IP of the acknowledgment messages from its child nodes. It is worth mentioning that a parent node could receive an acknowledgement message from a child node with a different parent node than itself. In this case, the messages are ignored. This process is used to identify while the query is propagating if there are child nodes available for a given node. If a node does not receive any acknowledgment, it continues the monitoring process by using a timeout. The timeout() refers to the process of counting time since the last package received. The done() refers to the restart of the state of the node. Meanwhile, error() refers to the process of not being able to send a message, which if it happens, it means that the node itself is out of the network range or a major

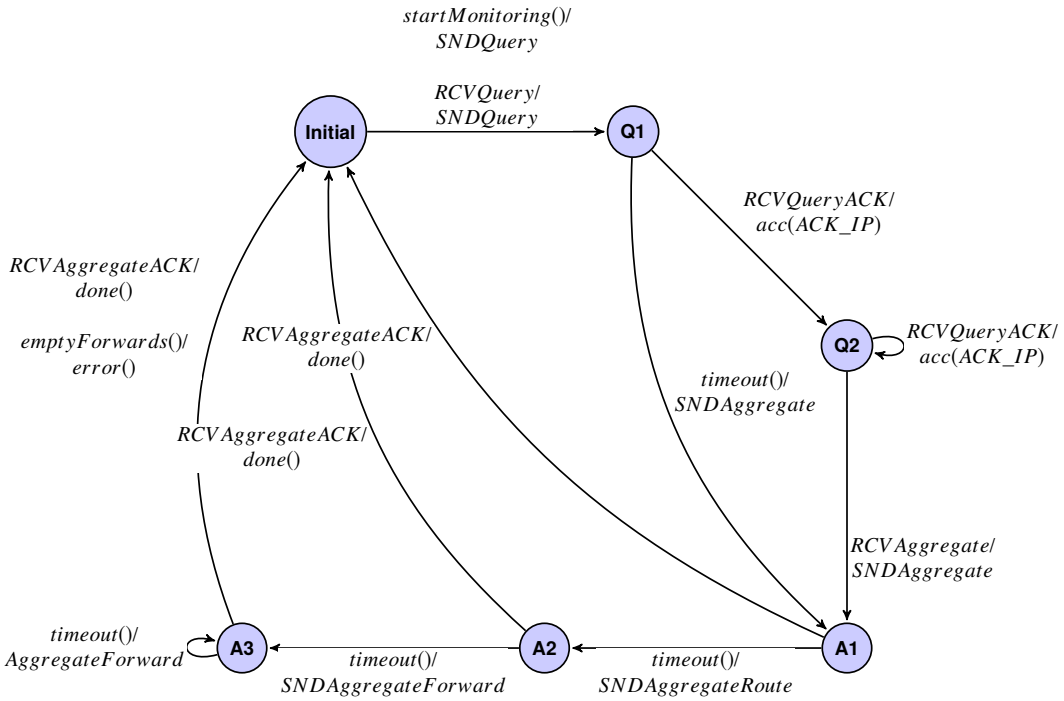


Figure 3.4 – Finite state machine of the protocol with mobility support

outage is happening with the network.

The input and output operations of the automaton are determined by sending (SND) and receiving (RCV) messages. The possible messages to be sent or received are the query, query ack, aggregate, and aggregate ack. The query message refers to the query itself and the base ground of the query state. For simplicity purposes, in the automaton, there is a distinction between the query and the query ack message. But in reality, they are meant to be the same package but received by a different node. This is discussed in Section 3.2.2. For example, if we refer to Figure 3.1c, we can see that node 1.1 is sending the query message to nodes 2.1, 2.2 and 0. For nodes 2.1 and 2.2, it means to receive a query message, but for node 0, given the properties transmitted in the package, then node 0 can compute that this message is the acknowledgment of his query message and that is why it is depicted as query ack. The aggregate messages refer to the aggregation process and the same principle applies as the query messages. The aggregate ack message is an aggregate message but received by a different node. In this case, the receiver node of the aggregate ack is the child node, as opposite as the query ack which the receiver is the parent node.

The first version of the protocol was designed without mobility support. This was done intentionally to first prove the behavior of it before adding the complexity to handle mobility. Therefore, the protocol definition with mobility support is depicted in Figure 3.4. The set of states is $Q = (Initial, Q1, Q2, A1, A3, A3)$ where $Initial$ is the initial state. The states $Q1$ and $Q2$ refer to the query states of the network. And the states $A1$, $A2$ and $A3$ refer to the aggregate states of the network. The internal operations of the automaton are the same as for the protocol without mobility support. Also, the input and output operations of the automaton are the same as for the protocol without mobility support with some additions. Besides the normal input and output operations, we specified two extra messages: the aggregate route and aggregate forward. The first one refers to the process of routing a message

through the network to the corresponding parent node in the VHT. As explained in Section 3.2.2, the idea is to make the hierarchical aggregation more robust through the addition of a gossip routing approach to route the package to the corresponding root node on the fly. Finally, when the parent node is not found, probably since it went off line due to an outage or something similar, it is forwarded to one of the nodes defined in the relay set. The relay set is populated by the grandparents and siblings nodes information.

The protocol with mobility support is below described in the Algorithm 1. The procedure is summoned when a packet (referred as payload) arrives on the input queue. From this point, it decides what to be done based on the state of the node. The behavior of the algorithm corresponds in general to the finite state machine from Figure 3.4.

Algorithm 1 Procedure that attends the incoming messages

Precondition: $state$ is defined and initialized as $INITIAL$, the $SEND*()$ functions trigger automatically a timer, the $sendAggregate()$ function aggregates internally results and observations

```
1: procedure ATTENDBUFFERCHANNEL(payload)
2:     type ← payload.Type
3:     switch state do
4:         case INITIAL
5:             if type = START || QUERY then
6:                 state ← Q1
7:                 SENDQUERY(payload)
8:             end if
9:         case Q1
10:            if type = QUERYACK then
11:                state ← Q2
12:                queryACKList ← append payload.Source
13:            else if type = TIMEOUT then
14:                state ← A1
15:                SENDAGGREGATE(parentIP, result)
16:            end if
17:        case Q2
18:            if type = QUERYACK then
19:                state ← Q2
20:                queryACKList ← append payload.Source
21:            else if type = AGGREGATE then
22:                state ← A1
23:                queryACKList ← remove payload.Source
24:                if EMPTY(queryACKList) then
25:                    SENDAGGREGATE(parentIP, results)
26:                end if
27:            end if
28:        case A1
29:            if type = AGGREGATE then
30:                state ← A1
31:                queryACKList ← remove payload.Source
32:                if EMPTY(queryACKList) then
33:                    SENDAGGREGATE(parentIP, result, observations)
34:                end if
35:            else if type = TIMEOUT then
36:                state ← A2
37:                SENDROUTE(payload)
38:            else if type = AGGREGATEACK then
39:                state ← INITIAL
40:                DONE
41:            end if
42:        case A2
43:            if type = AGGREGATEACK then
44:                state ← INITIAL
45:                DONE
46:            else if type = TIMEOUT then
47:                state ← A3
48:                SENDFORWARD(payload)
49:            end if
50:        case A3
51:            if type = AGGREGATEACK then
52:                state ← INITIAL
53:                DONE
54:            else if type = TIMEOUT & !EMPTY(relayList) then
55:                state ← A2
56:                relay ← poprelaySetList
57:                SENDFORWARD(payload)
58:            else if type = TIMEOUT & EMPTY(relayList) then
59:                state ← INITIAL
60:                ERROR
61:            end if
62: end procedure
```

There is an initial state (named *INITIAL* in our algorithm Line 4) to start the monitoring process or to continue the query process. From there and in state Q1 (Line 9), it can receive an acknowledgement for the query that will append the IP of the source to keep track of its children. Then it can loop (state Q2, Line 17) by storing all the children responding to the query of the node. When the nodes at the edge of the network do not receive any response during the defined timeout, they start the aggregation process (Line 13). This transition process is represented by an internal timeout message which sends an aggregate message. Then in the state A1 (Line 28), the nodes collect all the aggregate messages relying in the list it builds with the acknowledgements of the query messages. Once a node has collected all of the messages, it sends its own aggregate message with the aggregation of all the results of its children (Line 33). If one of the nodes delays in this process, a timeout process makes sure that the overall algorithm continues (state A1 - Line 35, state A2 - Line 46 and state A3 - Line 54). It is worth mentioning that if a node is not able to receive an acknowledgement from its parent, then it will fallback to the gossip routing process (state A2 - Line 42) or if that fails to the relay list process (state A3

-Line 50). This fallback process plans to deliver the aggregate message to the parent or to any of the grandparents in case the parent is offline (described in Section 3.2.2).

3.2.4 DHYMON: Enhancing Decentralized Monitoring Through a Multi-Root Nodes Approach

In the previous sections, we defined a hybrid monitoring architecture where a random root node is selected to start and conduct the monitoring process. These definitions, finite state machines and algorithm have shown promising background for a solid architecture but there is a common challenge regardless of the number of nodes, size of the network and speed of the nodes, which is the network fragmentation. Fragmentation leads to partial observations given that the nodes in the network are not in reach all the time. Assuming that all the nodes will always have a neighbor node is a strong assumption which can lead to over optimistic results from a non-realistic environment. Therefore, we have opted to tackle this issue by introducing this concept and aiming to improve the overall monitoring process with more realistic conditions. The network fragmentation directly affects the coverage of the algorithm. Depending on Algorithm 1, described in Section 3.2.3, each node is capable of being in three possible states:

- Non-covered: meaning that a node was not reached by the query procedure then not included in the aggregate procedure. This could be derived from fragmentation or availability.

- Partially covered: meaning that a node was reached by the query procedure but it was not able to send and acknowledge the aggregate packet. This could happen due to mobility.

- Fully covered: meaning that a node was reached by the query procedure. It was able to send the aggregate packet and it was acknowledged by its parent node.

Therefore, the coverage of our algorithm is defined by the number of nodes fully covered by the monitoring process. In other words, the coverage is equal to the number of results or observations that the root node receives at the end of the algorithm. The accuracy is the ratio between the coverage and the total number of nodes. It provides a metric to measure how accurate is our algorithm.

Relying in these definitions, we introduce the enhancement of monitoring based on multi-root nodes for a hybrid architecture. The aim is to improve the overall accuracy by covering more nodes. Aiming to maximize the coverage of the algorithm, we propose having multiple root nodes in order to cover more nodes who are left out due to the network fragmentation or any other communication problem due to mobility. For the purposes of this study, we have randomly selected two root nodes in order to improve the monitoring efficiency as a first approach. Each root node triggers a separate and parallel monitoring process, each one of them taking into account its own coverage. Then we analyze the coverage of the joint monitoring process to enhance the coverage and the accuracy of our algorithm over the network. Making reference to our coverage definitions, this means that each fully covered

node is taken into account. Then we will distinguish uniquely each fully covered node in any of the two root nodes. Our hypothesis is that through this process, we will be in a position to conclude with a higher accuracy. Due to this process, there are important premises worth mentioning:

- The overall traffic generation from our proposal will increase probably by a factor of two.
- Ideally, where all nodes are fully covered by both root nodes, there will be a coverage overlapping, which will not lead to any improvements.
- In the worst case scenario, the gain will be minimal, maybe including a couple of nodes to the overall result leading to a small gain in coverage and accuracy.
- In the best case scenario, the nodes will be able to cover more of the network. This will result in high coverage and accuracy gains. We believe that this is the motivating scenario, which will demonstrate the efficiency of our approach. This scenario could be more prominent in dense and dynamic networks.

3.3 Experiments

To support our architecture, we implemented and conducted multiple tests to provide the support for the implementation. We evaluate our proposal using an emulator built in-house described in Chapter 5. We started by defining the packet that our implementation used. Then we conducted multiple emulations for a different set of implementations. First, we test our implementation without mobility support in a non-mobile environment. This was done to do a first testing iteration in a simple environment. This allowed us to optimize and evaluate in a controlled manner the approach. Second, we test our implementation with mobility support in a mobile environment. For this case we tried to increment the number of emulations to achieve more data and more information about our approach. Finally, we test our implementation with mobility support for multi-root node approach. Once again, we tried to improve the number of emulations and the implementation to enhance accuracy, convergence time and the overall efficiency. Through this set of implementations, it allows us to ensure the accuracy of our approach in a diverse set of scenarios.

3.3.1 Packet Structure Definition

To have a successful communication, we need to define the monitoring packet. The packet works equally in both states of the network, query and aggregate states, but different information are sent depending on the state containing a set of common properties. It needs to contain some basic information in order to be useful for the following nodes and hops. The definition of such packet is done using json (Bray, 2014), given that it is one of the most used data interchange formats. For each state of the nodes, there is a set of transmitted values. The global values are:

- Type: the type of message being sent, the set of values are query, aggregate, aggregate_route, aggregate_forward.
- Parent: the IP address of the parent node of the node sending the message.
- Source: the IP address of the node sending the message.
- Destination: the destination IP, this for most of the cases should be the parent IP, unless the parent node is off line, then it will be a relay set IP.
- Gateway: the gateway IP, this is to support the routing which identifies the next hop of the packet.
- Timeout: the timeout in milliseconds to avoid infinite loops and to support the aggregate state.
- Timestamp: a unique identifier calculated with the IP of the source concatenated with the Unix time of the system.

Listing 3.1 – Decentralized approach, monitoring packet JSON definition

```
{
        "type": "query|aggregate|aggregate_route|aggregate_forward",
        "parent": "<parent IP>",
        "source": "<source IP>",
        "destination": "<destination IP>",
        "gateway": "<next hop IP>",
        "timeout": <time in ms>,
        "timestamp": "<sourceIP + UnixTime>",
        "query": {
                "function": "<monitoring function>",
                "relaySet": "[IPs of the source relay set]",
        },
        "aggregate": {
                "outcome": "<monitoring result>",
                "observations": <number of observations>,
        }
}
```

For the query state, the specific values transmitted are:

- Function: the function f to compute. We are considering the basic functions like CPU average usage, RAM average usage, or any other simple property.
- Relay Set: list of IPs for alternative paths. This list will contain at most three items. It should correspond to the parent, grandparent and great-grandparent node of a node. This list could be either smaller or bigger but we think that three items would suffice for the purposes of this approach.

For the aggregate state, the values transmitted are:

- Result: the result of the aggregation of the function f. This should be the aggregation of child nodes and the node itself.
- Observations: the number of observations aggregated. This value will be aggregated from the incoming aggregate packets.

The complete json definition of the monitoring packet can be seen in Listing 3.1.

3.3.2 Experiments without mobility support for non-mobile networks

The testbed consisted in an implementation of the protocol in the language Go [1], that was deployed on our emulator. The idea was to determine the convergence time, by which we mean the time it took from the moment that the monitoring started by the root node, to the moment that the root node was able to return a verdict. Along this, we also looked at the number of observations collected, number of packets sent and size of the packets per run. We defined two scenarios, one scenario with no mobility and another with low mobility. For this study, we are testing the implementation without the mobility support. This means we are relying on the automata described in Figure 3.3. Scenario 1 and scenario 2 consider the parameters of Table 3.1. For both scenarios the Mac Protocol is 802.11a with a data rate of 54Mbps. Each node had a range of ≈125m. Scenario 1 was designed to test the convergence time, the scenario 2 to prove that due to the high performance of the algorithm, we may monitor in a mobile environment without the mobility support.

The emulator was running on top of an AWS EC2 t2.large instance and Ubuntu 16.04. Versions in use were Docker 1.12.1, NS3.25 and Go 1.6.2. The containers were running as a base Ubuntu 16.04 LTS and IPv4. To collect the measurements we relied on the logs of the containers which were later parsed, exported to cvs and analyzed with R.

Results for non-mobile networks

For our first scenario, we collected the convergence time using a different root node as a starting point in each run. We decided to use a different root node selected randomly to prove that it will work independently of who the root node is. The network space is tied to the number of nodes. For 10, 20 and 25 number of nodes the network size is 500x500. For 40, 50 and 60 number of nodes the network size is 800x800. Finally, for 75, 80 and 100 number of nodes the network size is 1,000x1,000. It is also important to mention that we ran approximately 25 emulations for every different number of nodes for a total of 200 emulations. The results are summarized in Figure 3.5. The first remark we can make about the results is that it seems that there is a correlation between the number of nodes and the time it takes to converge. In the first segment of the graph, we can notice that from 10 nodes to 20

[1] https://github.com/chepeftw/Treesip

Table 3.1 – Parameters for the experiments without mobility support for scenario 1 (non-mobile network) and scenario 2 (mobile network)

	Scenario 1	**Scenario 2**
Number of nodes	10, 20, 25, 40, 50, 60, 75, 80 and 100	25
Network Space	500x500, 800x800 and 1,000x1,000	500x500
Network Positioning	Grid (100m apart)	Random
Running time	80s (init time 60s)	80s (init time 60s)
Emulation times	200	40
Mobility	-	RWP
Mobility Speed	-	2m/s and 5m/s

nodes, it increases significantly. This tells us that for small networks it is highly efficient, but as the number increases, so it does the convergence time. We can also notice a strange behavior between 20 to 40 nodes, which we determined it is due to some extreme cases that happened in these cases. This highlights the importance of doing multiple emulations to dismiss any extreme cases that can derive inaccurate results. Then, it seems that the average convergence time for 50 to 100 nodes, it stabilizes around ≈2.15s.

The number of packets sent, we theoretically assumed that for the static environments, the number would be twice the number of nodes. This can be deducted because every node sends the query message one time and an aggregate message one time as well. The variability of the number of packets will occur when the mobility support is added. For N nodes, the messages sent are $2 * N$ for static environments. The message size is variable, specially due to the relay set property in the package. The average message size is ≈151 bytes. This value was calculated using 600 different package sizes from a scenario of 100 nodes.

For non-mobile environments, the approach is able to cover the ≈100% of the network. This also means that the accuracy is ≈100%. This is expected due to the lack of mobility, but it is expected to vary in the mobile environment. Once again, this scenario is intended to test that the approach is able to converge and not to test the accuracy.

Results for mobile networks

For the second scenario, we collected the convergence time but also the amount of observations collected by the root node at the end of each run. This will tell us the accuracy and the success rate of the algorithm in mobile environments. For this scenario, we tested for only 25 nodes and a network size of 500x500. The mobility model in use is the random waypoint mobility model (RWP), using two different speeds of 2m/s and 5m/s. For each speed, we ran 20 emulations to ensure the significance of the results, in total 40 emulations. The results are summarized in Table 3.2. We can observe that the nodes converge about the same amount of time that they do in a static environment. This resemblance

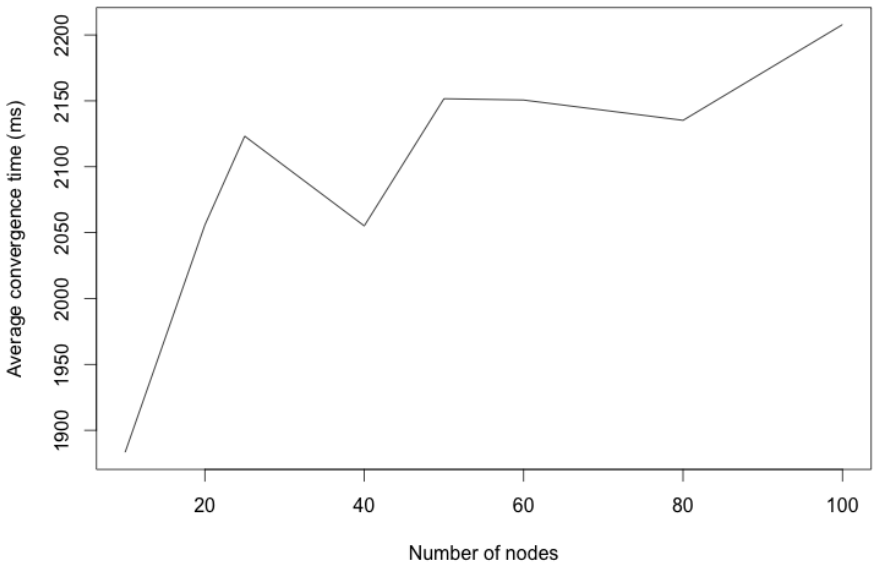


Figure 3.5 – Experiments without mobility support for scenario 1 convergence results

is due to the timeout of the edge nodes. In both scenarios, the timeout value is the same, and therefore it is what causes the big delay for the convergence time. The timeout value was chosen arbitrarily.

We observe that the approach would converge but without all the possible observations on the network. Aside from that, the higher the speed of the nodes then the lower the success rate is. By success rate, we define if the monitoring process was able to converge. Even though, the results might seem discouraging for mobile scenarios, but they are really promising. The algorithm has proved in static environments that is capable of converging really fast, even though is not using the mobility support, suggesting promising results when we include the mobility support.

Once again, the tests for our approach without mobility support, have proof that our work is capable to work when integrating gossip and hierarchical approaches into a hybrid proposal. It also proved that it can converge in a reasonable time. The accuracy in the static environment is $\approx$100%, which it was expected and it is not interesting to analyze. The accuracy for the mobile environment is different, and this is an interesting result since it converges although is working without mobility capabilities. Although the promising results, there are questions and drawbacks from the approach that we can discuss. The first one refers to the root node, for the moment we have chosen it randomly. But we intend to study the selection of the root node. It could be based on location, energy, computing power and other parameters, or to be an autonomous process, proactively or reactively, or a manual process. Besides we plan to introduce the mobility support and enhance the testbeds to this specific cases. In particular, we expect to assess the performance in terms of convergence of communication packets with a high node speed. We believe that the proposal is very promising, but there are multiple areas to define and enhance. Another area of interest, is to handle the monitored property after the time window. So far this process is calculated using a hybrid algorithm, but it could be stored in all nodes or in a coordinator. We intend to analyze this options into our proposal, to maximize the options and scenarios to cover with our approach. We also intend to consider more complex functions in monitoring the MANETs interoperability. For this we need to define an optimal solution to propagate a more complex function through our query mechanism.

Table 3.2 – Experiments without mobility support for scenario 2 results

	Speed 2m/s	**Speed 5m/s**
Average convergence time	2,018.06	2,123.63
Average observations	21.68	10.25
Accuracy	≈80%	≈40%

3.3.3 Experiments with mobility support for mobile networks

The testbed consisted in an implementation of the protocol in the language Go named Treesip[2]. The idea was to determine the convergence time, by which we mean the time it took from the moment that the monitoring started by the root node, to the moment that the root node was able to return a verdict. Along this, we also looked at the number of observations collected, number of packets sent, size of the packets per run and the accuracy of the measurement. The accuracy is defined as the ratio between the number of observations and the number of nodes. We defined four scenarios with mobility. We compare different number of nodes, space, speed, mobility pattern and routing protocol. The first scenario runs with a defined speed of 5m/s but different number of nodes and dimensions and considers the parameters of Table 3.3. The second scenario runs with the same values as the first scenario, but with three different mobility speeds 5m/s, 10m/s and 15m/s. The third scenario runs with the same values as the first scenario, but with two different mobility patterns. Finally, the fourth scenario runs with the same values as the first scenario, but with different routing algorithm as a fallback process. In the first three scenarios we used our proposed gossip fallback mechanism defined in Section 3.2.2, but in this we also test OSPF to compare the differences. For this study, we are testing the implementation with the mobility support. This means we are considering the automata of Figure 3.4. For all scenarios the MAC protocol is 802.11a, the application protocol is UDP and with a data rate of 54Mbps. Each node had a range of ≈125m (theoretically by default in NS3 is 200m). All scenarios were designed to test the convergence time and accuracy in a mobile environment. Given the mobility aspect of the scenarios, we assumed that it is difficult to achieve a fully accurate measurement. But the goal is to maximize the accuracy through the proposed architecture.

The emulator was running on top of an Amazon EC2 instance of type t2.medium and Ubuntu 16.04 LTS. Versions in use were Docker 1.12.1, NS3.25 and Go 1.6.2. The containers were running as a base Ubuntu 16.04 LTS and IPv4. To collect the measurements we relied on the logs of the containers which were later parsed, exported to cvs and analyzed in R for further analysis.

Results varying the number of nodes

For our first scenario, we collected the statistics using a different and random root node as a starting point in each run. We decided to use different root node to prove that it works independently of the start root node. The parameters for this scenario can be seen in Table 3.3. The idea of this scenario is to test the approach by including the mobility support and testing with a mobile network. The results

[2]https://github.com/chepeftw/Treesip

Table 3.3 – Base parameters for the experiments with mobility support for all scenarios

Parameter Name	Value
Number of nodes	20, 25, 40, 50
Network Space	400x400, 600x600
Network Positioning	Random
Emulation runs	100 (per number of nodes)
Emulation time	60s each with an init time of 40s
Mobility	Random Waypoint Model
Mobility Speed	5m/s

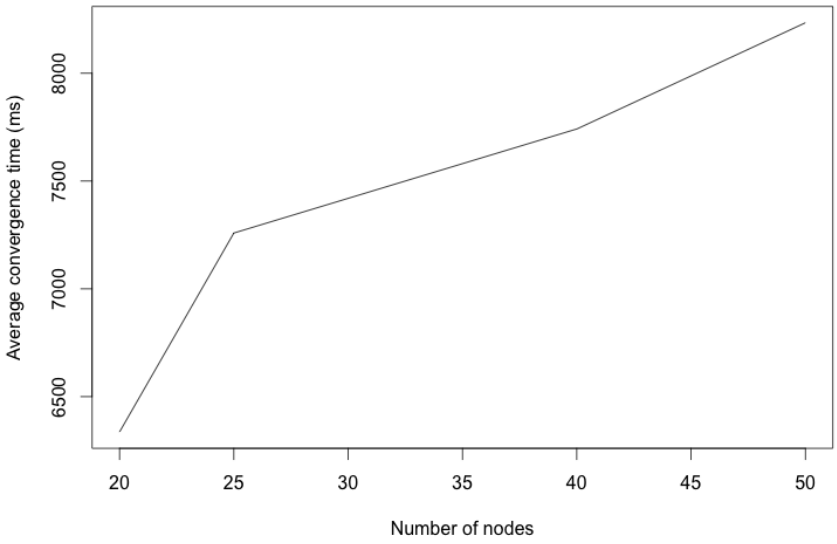


Figure 3.6 – Experiments with mobility support for scenario 1 convergence results

are summarized in Figure 3.6. We can point out that there is a clear correlation between the number of nodes and the time it takes to converge. It is important to note that these times are subject to the defined timeouts for each run. We based our timeouts in empirical data based on the number of nodes on different emulations. Trying to find a safe value that will allow the network to converge. In overall, the average converge time it is approximately ≈7.51s. Once again, the main reason for the convergence time is the timeout value, the higher this value is the higher the convergence time it is.

About the number of packets sent, we empirically assumed that it would increase depending on the number of nodes given the gossip aggregation routing messages. For 20 nodes the average per run was 89 packets, for 25 nodes it was 138 packets, for 40 nodes it was 226 packets and for 50 nodes it was 280 packets. The message size is variable, specially due to the relay set property in the package or for the properties transmitted for routing information. The average message size for all runs is approximately ≈98 bytes. The results collected by the monitoring process are approximately ≈80% accurate. Compared to the results shown in the previous section, there is a significant increase in the accuracy for a network moving at 5m/s. This is due to the mobility support which supports the VHT and the aggregation procedure. On the other side, the convergence time increase, but this happened due to the fact that we increased the timeout.

Table 3.4 – Experiments with mobility support for scenario 2 results

	5m/s	**10m/s**	**15m/s**
Average convergence time	6,336.12	6,245.54	9,419.90
Average observations	20	19.8	18
Accuracy	≈80%	≈79%	≈72%

Table 3.5 – Experiments with mobility support for scenario 3 results

	Random Waypoint	**Random Walk**
Avg. convergence time (ms)	6,336.12	8,930.13
Avg. observations (# nodes)	20	18
Accuracy	≈80%	≈72%

Results varying the node speed

For this scenario, we utilized 25 nodes, a 350x350m space and the random waypoint mobility model. We tested the speeds 5m/s, 10m/s and 15m/s. For each one, we ran the simulations 25 times. The results are summarized in Table 3.4. We assumed that the converge time might remain stable, but the accuracy could drop for a higher speed. We can observe that the nodes converge about the same amount of time in the first two scenarios but for the third one increase significantly (difference of ≈3.2s). The difference is due to the added difficulty to route the monitoring aggregation when the hierarchy broke physically. This leads to multiple attempts to route different packets through the network which eventually causes an increase in the convergence time. Contrary to our preliminary expectations, the accuracy remained approximately the same for 5m/s and 10m/s, and for 15m/s it slightly decreased. The algorithm is capable of converging with nodes moving at different speeds with a stable accuracy. This shows promising results for higher mobility scenarios.

Results varying the mobility model

For this scenario, we utilized 25 nodes, a speed of 5m/s and a 350x350m space. We used the mobility patterns from NS3 of random waypoint model and random walk model. For the random walk model, we used a stop time of 2s. For each mobility pattern we ran the simulations 25 times. The results are summarized in Table 3.5. We can observe that the convergence time has a difference of ≈2.5s, the average observations have a difference of 2 nodes and consequently the accuracy is different by ≈8%. For this specific case, although the convergence time increased between the mobility models, the accuracy was no heavily impacted. This assures us that the mobility pattern can have a direct impact on the convergence time due to the network fragmentation. In overall and in our opinion, the results are fairly similar. But most importantly, it shows that it works in different mobility patterns whatever the properties are affected.

Table 3.6 – Experiments with mobility support for scenario 4 results

	Gossip	**OLSR**
Average convergence time	6,336.12	4,272.65
Average observations	20	14
Accuracy	≈80%	≈56%

Results varying the routing fallback procedure

For this scenario, we utilized 25 nodes, a speed of 5m/s and a 350x350m space. In this simulation, our intention is to compare the performance of our gossip routing approach and the protocol OLSR based on the study of Jacquet et al. (Jacquet et al., 2001). For each approach we ran the simulations 25 times, giving one minute for OLSR to have enough information. The results are summarized in Table 3.6. The convergence times are slower in our approach but more accurate, meanwhile in OLSR is faster but less accurate. This results derives an interesting discussion. If we use OLSR, our approach becomes dependent on the routing layer, but at the same time it delegates this responsibility and it allows our approach to focus on the monitoring process. If we rely on our gossip routing approach, the solution becomes independent but it adds more complexity. So there is a clear trade-off that should be considered. After analyzing the logs and the scenarios we concluded that some scenarios were not so accurate because of the node reachability. This is something we intend to study in depth in future works.

These set of emulations show that our approach is capable to work in scenarios with different number of nodes, network sizes, speed, mobility model and routing protocol. It is also worth mentioning that we have conducted a significant amount of emulations. For scenario 1 the emulation was ran 400 times. For scenario 2 the emulation was ran 300 times. For scenario 3 and 4 the emulation was ran 200 times per each. This gives us the certainty to assure that our approach works repeatedly and not only works in a specific and controlled scenario. Regarding the results, it is fair to assume that the timeout has a direct impact in the convergence time and it needs to be analyzed more in depth to speed up our approach. On the other side, the accuracy seems to stabilize in ≈80%. Regardless of the variations it did not go higher than that. This is due to the timeout as well, some of the results are considered lost when the timeout is reached. This behavior is inherited from the gossip algorithms, specifically the random gossip algorithms. This behavior is also caused due to the network fragmentation. The studies in the literature which operate in mobile environments, have to struggle with this MANET property of mobility and due to this the accuracy becomes an indicator for how well the approach performs. This highlights the necessity to improve our approach to increase accuracy and decrease the convergence time.

Based on the work of Battat et al. (Battat et al., 2014), there are some works that can be mentioned to keep in perspective our work. In the gossip-based categorization, we can discuss Mobi-G (Stingl et al., 2014) which is inspired by Gossipico (Van De Bovenkamp et al., 2012) for static networks. This is an algorithm to calculate the average, the sum or the count of node values in a large dynamic network. The combination of two mechanisms, count, and beacon, provides the nodes of the network

counting efficiently and quickly. It is designed for urban outdoor areas with a focus on pedestrian that moves around by foot. It can provide accurate results even for fluctuating attributes. It also can reduce the communication cost. It does not suffer from extended range connectivity, but the accuracy decreases for an increasing spatial network size. In the hierarchical categorization, BlockTree (Stingl, Gross, Nobach, et al., 2013) proposes a fully decentralized location-aware monitoring mechanism for MANETs. The idea is to divide the network into proximity-based clusters, which are arranged hierarchically. Each cluster or block aggregates the data respecting a property. The algorithm requires that all nodes from the same cluster or block be reachable within one hop. Even though the excellent performance, the average power consumption increases directly proportional to the spatial network size or node density.

3.3.4 Experiments for multi-root node for mobile networks

We evaluate our proposal using an emulator built in-house described in Chapter 5, but relying on the second stage which is the AWS EC2 orchestrator. The orchestrator parses the logs from the Docker containers per emulation, extracts the information and centralizes it in a MongoDB server for further analysis. The testbed consisted in an implementation[3] of our proposal in the language Go (v1.8). The idea was to identify primarily the convergence time and the accuracy. The convergence time is described as the time it takes from the moment that the monitoring started by the root node, to the moment that the root node was able to return a verdict.

We define two different test suites. First, we analyze the timeout selection. In the past experiments, a random timeout of 1800ms was used and we concluded that this value affects directly the convergence. However, we think we need to analyze this further to provide a better value or a better estimation for further works. Therefore, we simulate the single root node varying the timeout using different number of nodes. This allows to find a pattern for a better timeout maximizing the convergence time. Second, we study the convergence and accuracy of the multi-root node approach. Our hypothesis is that using more than one root node, we will be in a position to capture more node information, even though we have overlapping results. For this case, we simulate multiple number of nodes using different network sizes varying the speed of the nodes and finally comparing this to the single root node results varying the same parameters.

For all scenarios, the MAC protocol is 802.11a, the transport protocol is UDP with a data rate of 54Mbps. Each node had a range of ≈125m. We rely on the random waypoint mobility model, using a pause time equal to 0 providing therefore continuous mobility. All scenarios were intended to test the convergence time and accuracy in a mobile environment. All simulations have an initial configuration time to randomize the position of the nodes. The idea is for the nodes to shuffle from their original location, regardless of the fact that they are located with a random pattern by NS3. All of this is done to enhance the trustability of the results by using random values to ensure that the algorithm works in any given scenario and that it is not tied by any emulation parameter. Given the node range, we decided that every 100m×100m of network size should have at least 2 nodes, then based on this requirement, we calculated the network size and the number of nodes. The number of nodes also determined the type of

[3]https://github.com/chepeftw/Treesip

Table 3.7 – Number of nodes, network size and server type relation for the experiments for multi-root node approach for all scenarios

Number of nodes	Network Size	AWS EC2 type
20	316x316	t2.small
30	387x387	t2.small
40	447x447	t2.medium
50	500x500	c4.large
60	548x548	c4.large

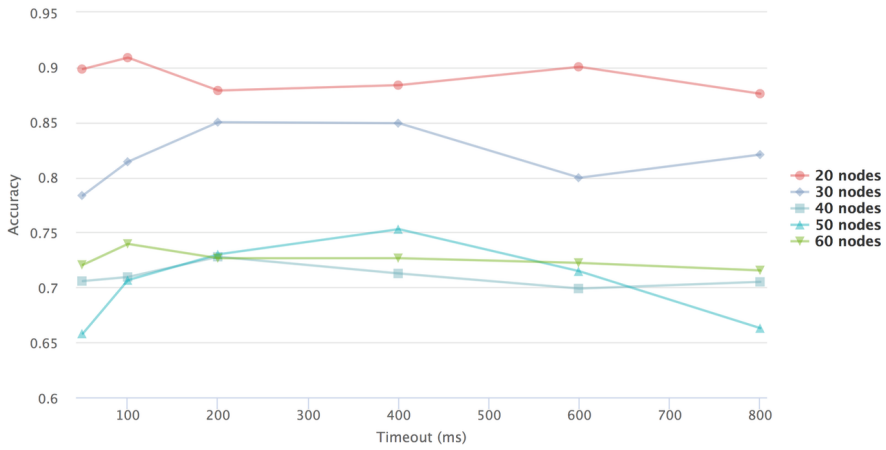


Figure 3.7 – Timeout vs accuracy for nodes at 2m/s for the experiments for multi-root node approach

Amazon EC2 instance where the emulation was running. This was due to the fact of the computation power required by NS3 as the number of nodes and number of packages to simulate increases. The parameter relationship between number of nodes, network size and Amazon EC2 instance type is summarized in Table 3.7. Then, for each separate arrangement of parameters, the emulations ran for 100 cycles to ensure stable and consistent results.

The versions in use were Ubuntu 16.04 LTS, Docker 17.03.1-ce, NS3.26 and Go 1.8. The containers were running as a base Ubuntu 16.04 LTS and IPv4. To collect the measurements, we relied on the logs of the containers which were later parsed, exported to MongoDB and analyzed with the help of Python and JavaScript.

Experiments and Results for the Timeout Selection

As mentioned before, in the previous experiments, random timeout value of 1800ms had been used, which based on our tests we were able to conclude that it affects in direct proportion the convergence time of our algorithm. Therefore, we decided to analyze this further by varying the number of nodes, network space, node speed and timeout to determine a pattern or a better value. The results of this experiment for different number of nodes moving at different speeds of 2m/s and 10m/s can be seen respectively in Figure 3.7 and Figure 3.8. Each data point in each graph represents 100 emulation runs using the same parameters, this means that each graph represents 3,000 emulation cycles.

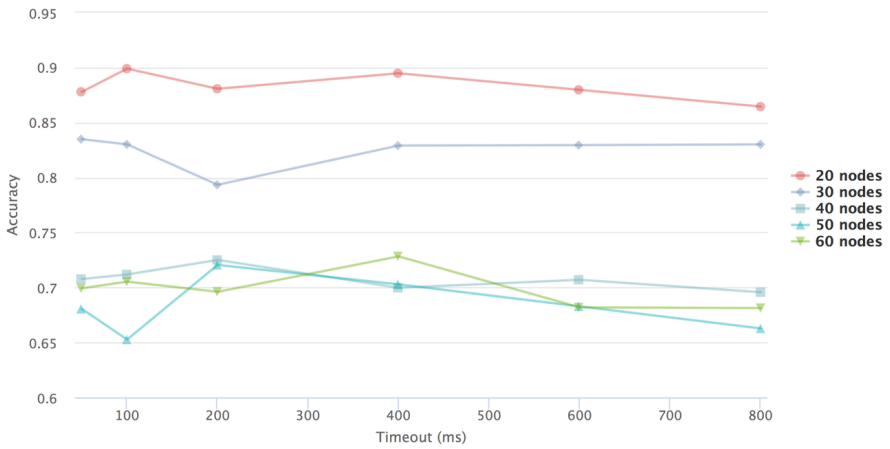


Figure 3.8 – Timeout vs accuracy for nodes at 10m/s for the experiments for multi-root node approach

Table 3.8 – Accuracy and convergence results by timeout for the experiments for the timeout selection for multi-root node approach

Timeout	**Accuracy**	**Convergence**		
		Average	**Minimum**	**Maximum**
50	0.77	2,523.66	55	16,253
400	0.76	3,080.23	403	22,564
200	0.76	2,952.15	202	21,262
600	0.76	3,503.95	603	17,372
100	0.76	2,766.03	102	18,850
800	0.76	3,920.75	804	20,477

The results in Figure 3.7 show that the most prominent and promising timeouts are 200 and 400. But in Figure 3.8, it seems that 50 and 200 could be the most prominent and promising in general. We may conclude from these graphs that there is no pattern and that the underlying problematic could be more complex that we previously thought. If we analyze this experiment from another perspective (Table 3.8), we can group the results by timeout from the different node speeds and different number of nodes sort by accuracy. Each row of this table represents 2,500 cycles of emulation. Also the convergence column shows the approximate average value, followed by the minimum and maximum value between the parenthesis. We can observe three interesting results. The empirical assumption would be that the timeout and the accuracy could have a direct correlation, but from the data we can see that this is not the case. Based on our past studies speculations, it shows a direct correlation between the timeout and the convergence time. We can observe different average convergence times, but if we take a closer look at the minimum value of the samples, it shows that in the best case scenario, the process would take almost the time of the selected timeout. And the last remark is that, depending on the mobility, it seems that it can last 20 seconds to try to converge. This shows that the worst case scenarios, it will take a lot of time due to the gossip approach but it will converge nonetheless. In these scenarios, the average routing messages increase significantly.

Results for the Multi-Root Node Approach

From our preceding experiments, we have seen that the general accuracy of the monitoring process is approximately ≈80%. In this study, we intend to enhance the accuracy by introducing a second root node. As defined in Section 3.2.4, the coverage is based on the nodes which were able to be fully covered. It means that different monitoring processes performed by two different root nodes could have a different set of fully covered nodes. This difference could be none, meaning that the two processes were identical which is the theoretical best case scenario, but not realistic. This is more like the theoretical best case scenario, but in real emulations, we expect this to not occur often. The difference could be more than one node and then could lead to a major enhancement. To calculate these sets and differences, we analyzed through the logs which nodes were fully covered by which monitoring process. This provided two distinct coverage lists that we crosschecked. By doing this, we concluded on a list with the nodes covered by the two processes. The results are summarized in Figure 3.9. Every data point of every graph represents 100 emulation cycles, therefore every graph contains 2,000 emulation cycles.

To show the improvements of our proposal, we ran again, under the same conditions, the emulations for single-root node approach and we illustrate them in the same graph as their multi-root node approach counterpart. We did this comparison by using two different timeouts and different node speeds to ensure that the enhancements are consistent. In the graphs, value t refers to the timeout.

In all the graphs, we can note that there is a consistent enhancement between the single-root node and the multi-root nodes approach. For this, we calculated the difference between all the corresponding data points and calculated that the average enhancement is ≈12.87%. The minimum is 0.06 and can be seen in Figure 3.9d in the 400ms timeout for 20 nodes. The maximum is 0.24 and can be seen in Figure 3.9d in the 400ms timeout for 50 nodes. The general accuracy for all the multi-root nodes experiments is ≈90.06%. From these results, it is fair to say that the multi-root nodes approach increases the general accuracy by ≈12% (based on 10,000 emulation runs). This is an interesting result since shows a solution to network fragmentation by covering the network from different nodes at the same time in a monitoring scenario. This also opens the question about how many root nodes would be the ideal number for providing efficient monitoring and minimizing the network traffic. It is worth mentioning that if we were to consider that the number of root nodes would be equal to the number of nodes in the network, this would be no longer a decentralized approach but a distributed approach.

To analyze the convergence time of the monitoring process for the 200ms timeout, we can refer to Table 3.9, and for the results for timeout of 400ms, we can refer to Table 3.10. Each row in the table represents 500 emulations cycles. Also the convergence and tree depth columns show the approximate average value, followed by the minimum and maximum value between the parenthesis. We can note that there is a direct correlation between the number of nodes and the time it takes to converge. This is an expected behavior and it is also reflected in Section 3.3.4. The overall average using a timeout of 200ms is ≈3,066.98. We can observe that the minimum convergence time is determined by the timeout. These results give a great view of the performance of our algorithm, suggesting it can converge in a relatively fast time while providing a great accuracy. Regarding the tree depth, this confirms the hypothesis from Section 3.2.2. Therefore, we can state that the approach has an approximate of 4 to 6

Table 3.9 – Multi-root convergence results for 200ms timeout for the experiments for multi-root node approach

Nodes	Convergence			Tree depth		
	Average	Minimum	Maximum	Average	Minimum	Maximum
20	1,307.37	212	20,361	4.03	2	8
30	1,856.72	223	18,907	5.10	2	11
40	3,685.19	220	16,707	5.16	3	9
50	3,657.33	206	13,687	5.88	4	16
60	4,822.39	202	15,583	5.93	4	11

Table 3.10 – Multi-root node convergence results for 400ms timeout for the experiments for multi-root node approach

Nodes	Convergence			Tree depth		
	Average	Minimum	Maximum	Average	Minimum	Maximum
20	1,498.07	414	9,821	3.99	2	8
30	2,141.12	421	15,450	5.18	3	11
40	4,395.58	414	15,896	5.16	3	9
50	4,470.63	404	27,003	5.87	4	10
60	5,404.73	404	17,409	5.89	4	9

levels of depth in the VHT. Note that there are scenarios where the VHT can reach up to 16 levels and nonetheless it will be able to converge. And in the best case scenario, it can rely on only 2 levels, which theoretically are more than enough to cover the majority of the network. It is important to note that there are no scenarios with 1 level. This proves that the initial configuration time is important since it provides accurate scenarios.

In this experiment, we also analyzed the amount of time that our implementation took to handle each packet. This time represents the duration of the implementation of our algorithm taken to analyze the packet, to change state and to respond when necessary. The average time is 93,887.43ns, this means 0.09ms. The minimum time is 26,974ns and the maximum time is 1,540,137ns. We are aware of the fact that this time is dependent of the hardware where the implementation is running. This time means that our implementation is playing a minor role in influencing the convergence time.

Results for the Gossip-based Routing Fallback Procedure

From the multi-root nodes experiments, we analyzed how often was the routing layer in use and how it varied depending on the different number of nodes. We analyzed the results by node speed as well but interestingly the results were stable. Meanwhile, with different number of nodes, there was a big variance. The results are shown in Table 3.11. Each row represents 500 emulation cycles. It can be observed that the more the number of nodes, the more the messages are routed through the fallback mechanism. This is congruent with the purpose of the routing layer, given that it reacts as a fallback

Table 3.11 – Routing layer usage based on the number of nodes for the experiments for multi-root node approach

Nodes	Messages Sent	Messages Received
20	8.94	8.22
30	15.75	13.92
40	35.70	29.81
50	40.58	33.55
60	48.63	39.71

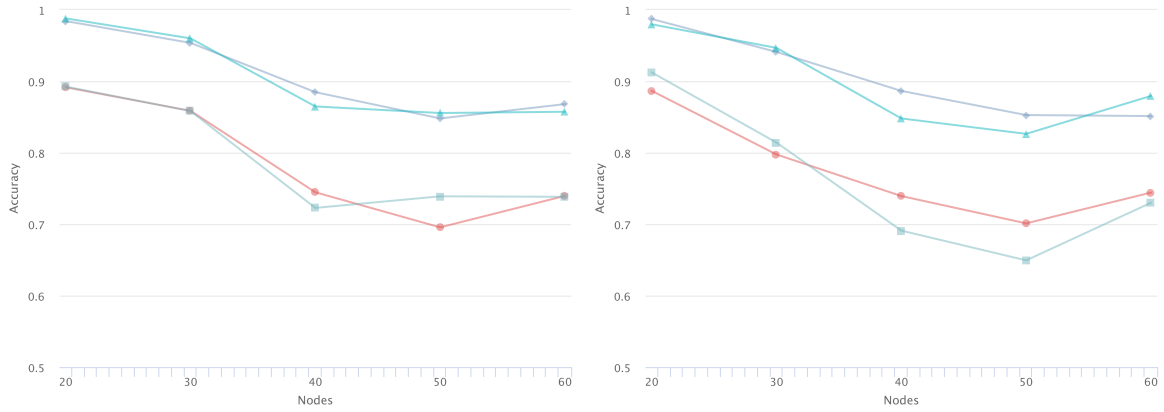
mechanism to make the communication possible between a parent node and a child node in the VHT. Our algorithm takes into account the fact that the more the nodes, the more the network fragmentation will occur. It is also important to notice how the delivery ratio drops as the number of nodes increases. These are interesting results since they point out the importance of this fallback mechanism and a feasible point for enhancements.

3.4 Summary

In this chapter, we presented our hybrid-based approach for monitoring decentralized networks which is a combination of gossip-based and hierarchical-based algorithms. The algorithm has been thoroughly defined which helps to be more confident in the mathematical background and the implementation results. The algorithm is constructed on top of two major procedures, the query, and the aggregate. The gossip-based approach is applied to the query procedure to disseminate the query efficiently. Once the query is propagated, the network changes to the aggregate procedure. Besides, with the help of a time-based hierarchical approach, the computation of a global property is achieved. We first define a single-root node approach. Afterward, we optimized our proposal by introducing the concept of the multi-root node approach. This allows the monitoring architecture to run various monitoring process simultaneously. We analyzed the case for two root nodes and obtained interesting results. The approach was evaluated by three campaigns of experiments which led to promising results.

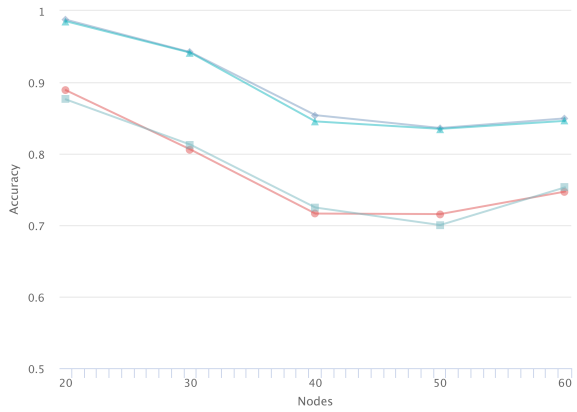
Although the efficiency of the approach, we started discussing the possible following enhancements for our proposal. From this, the first intrigue we had was how much the approach could be improved by increasing the number of root nodes. We agreed that devoting a study to the case for three root nodes was not appealing at all. But from this we hypothesized that if one root node delivered promising results, then two root nodes improved by $\approx$10% the accuracy, therefore the more root nodes in the network, the more accuracy gain. The result from this was that to maximize the accuracy, then all the nodes in the network had to be root nodes. We quickly observed that if the number of packets generated by our approach increased more than the double from single-root node to the multi-root node experiments, then the more root nodes meant more traffic. This, of course, was not what we wanted. So a new approach following the same line was needed. Surveying the literature did not motivate anything in particular, but when observing a novel technology in a different field, we quickly saw a possible solution to our problem. The blockchain technology has driven a lot of attention in the last

years, for some is called the next Internet, others call it the most revolutionizing technology of our times. From the other side, it has also been called a fiasco for energy efficiency, and the news oscillates among different opinions. It is worth mentioning that this fluxing news are due to the cryptocurrencies. But academics agree that the underlying technology, the blockchain technology, could derive new solutions. From this, we decided to analyze the technology, understand it, and evaluate the feasibility of an approach that could couple with our requirements.

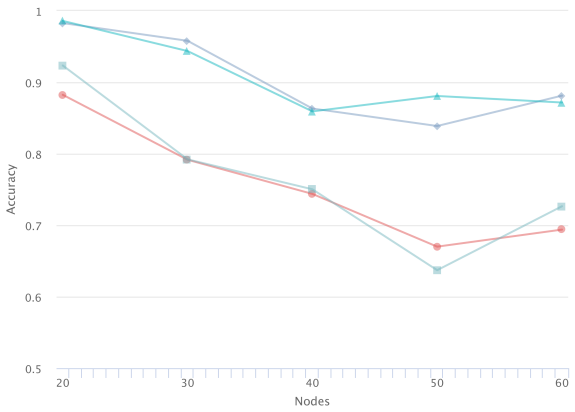


(a) Node speed 2m/s

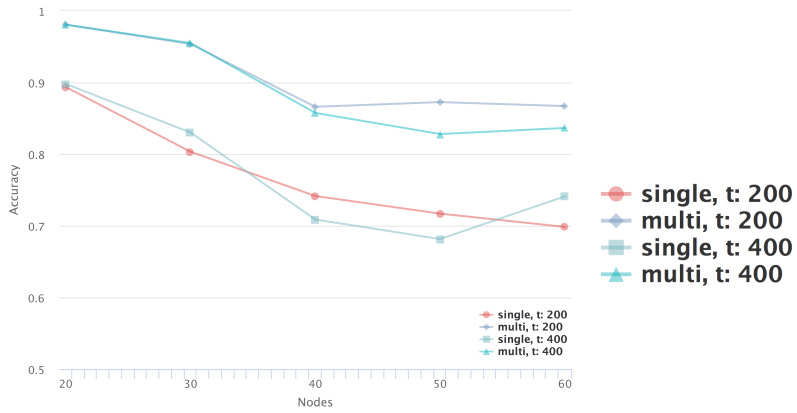
(b) Node speed 4m/s



(c) Node speed 6m/s



(d) Node speed 8m/s



(e) Node speed 10m/s

Figure 3.9 – Accuracy vs number of nodes with single-root node and multi-root nodes for different timeouts and different speed.

4

Distributed Monitoring

Contents

As stated in Chapter 2, most of the known monitoring approaches in the literature state that accuracy is not reliable due to the dynamic properties of the MANET. Network fragmentation has a harsh impact on the accuracy of a decentralized and distributed approach. Given that the information is spread through the network then if a subset of nodes of the network fragments, there is a probability that the data for the monitoring result is fragmented as well. Therefore, there have been many studies that try and have successfully increased and maintained the accuracy of the approaches, but most of the time the accuracy decreases with the number of nodes or density of the network. We observed this behavior in our previous experiments in Chapter 3. Although our experiments were promising, the accuracy would decrease as the number of nodes increases. Regardless of this, we observed that the

multi-root node approach offered an interesting perspective. The first question is how many root nodes would we need to improve the accuracy without creating excessive network noise. Our first idea was to run emulations with multiple root nodes ranging from two up to the number of nodes in the network. Instead of going this way, we noticed that if all the nodes in the network are root nodes, this means that the approach becomes fully distributed. Due to this, we decided to explore the option of a distributed approach. We immediately realized that making all participants of the network root nodes would have a significant impact on the network medium utilization, and therefore a new approach was needed.

Looking at a survey from NASA (Goodloe & Pike, 2010), which looks at monitoring distributed systems, they state that in such kind of systems there is a need to monitor multiple points in the network, and then reach a consensus among the different information to have useful and processed data. The consensus is primarily needed to reach a global and shared view among the nodes making sure that the agreed information is complete and accurate. Faulty nodes or incomplete information is a known problem in distributed systems. The same problem arises in the MANET context given the dynamic behavior of the nodes and the mobility. To overcome this unintended uncertainty of the nodes, a consensus algorithm helps to tackle this problem in a more efficient approach. This led us to believe that a distributed system combined with a robust consensus algorithm could provide a robust monitoring mechanism. From a different context, we observed a novel and similar solution being applied in a different domain, the blockchain. Therefore, we took this as an inspiration bedrock and from there we started to build a proposal that tackles the problematics for monitoring a MANET. A blockchain relies on different concepts, a distributed ledger, a consensus algorithm, cryptographic mechanisms and network protocol. Which somehow resonates with the needs we observe for monitoring a MANET.

The main contributions of this chapter are:

a) The proposal of a mathematical definition that models local results for a single node and global results for a group of nodes or the whole network. These results are the responses to a query definition that is propagated to the network. Finally how these are integrated into a global result definition. These definitions help us to provide a strong foundation for our monitoring architecture.

b) The proposal of a distributed monitoring architecture that relies on multiple points of observations being the nodes. It provides a consensus mechanism that allows the architecture to aggregate and provide a more meaningful result by analyzing the data from a bigger perspective. All results are stored integrally to ensure the trust of the data. Finally, we provide an appropriated mechanism to free stored results to avoid resource consumption problems.

c) Finally we evaluate our result in a network emulator built in-house which is described in detail in Chapter 5. This emulator allowed us to run thousands of emulations varying parameters like the number of nodes, network size, node density, speed, mobility pattern and timeouts. As a result, we were able to generate data and make an in-depth analysis of it by looking at average, maximum, minimum and standard deviation values for accuracy, convergence time, number and size of packets.

This chapter is divided as follows: In Section 4.1, we give the basic concepts and notions about

distributed monitoring and consensus algorithm, and how that influences our architecture. Then, in Section 4.2, in the first half, we define our mathematical models for the local and global result. Also, in the second half, we define our architecture based on the mathematical models. Finally, in Section 4.3, we describe our large campaign of experiments where we highlight the importance and effectiveness of our approach.

In this chapter, we expose the work which was submitted in the following journal: (Alvarez et al., 2018).

4.1 Preliminaries

Network monitoring is an extensive field of interest. It is described, as mentioned in Section 3.1, by Cormode as "a number of observers making observations and wish to work together to compute a function of the combination of all their observations". For the purposes of this chapter, the function can be simple or complex, therefore can model the average CPU of all nodes or the integrity of communication by checking the checksum of packets in all nodes.

We have to cite the works for the formalization of functional protocol properties such as (Mouttappa, Maag, & Cavalli, 2013; Lalanne & Maag, 2013; Che, Maag, Tan, Tan, & Zhou, 2015; Abid, Othman, & Shah, 2014). These works have also been an inspiration for the herein proposed approach in terms of models or formal definitions. But they are mostly limited to local validation made by local nodes and do not allow a distribution of the verification process. Nevertheless, they permit the definition of complex properties or system rules with causalities in MANET, which is worth mentioning.

4.1.1 Distributed Monitoring

The classification of the monitoring process has been studied in (Battat et al., 2014; Cormode, 2011). For the purposes of this chapter, we consider the one major type: distributed. As stated by (Cormode, 2011), the problem of distributed monitoring can have various "trivial" solutions, e.g., centralizing the information. The centralized type of monitoring is when all the nodes report their observations to a central entity, named centralizer or coordinator. The distributed approach deals with networks where there is no centralized entity and the data or computing of it is distributed across all nodes. This implies that the network by itself needs to achieve a global view of a property of the network. When the property to monitor is monotonic, it can be easier to achieve a global view. However, when the property to monitor is a non-monotonic, which are dependent on more information and non-deterministic, the solutions also tend to be more complex. A solution for approaches that need to be considered by monotonic and non-monotonic properties is defined as the geometric approach by (Sharfman, Schuster, & Keren, 2007). Sharfman et al. state that a global view can be achieved by breaking down the function into properties or conditions that can be checked locally. Another aspect of distributed monitoring is sampling, where there have been multiple studies, and they focus on an infinite window or sliding windows. The idea of infinite windows refers to that all nodes send all their

information to a coordinator, which can consume a significant amount of resources. On the other side, sliding window refers that we do not need to see events from all history but just recent events. The work of Goodloe et al. (Goodloe & Pike, 2010), states that another important aspect of distributed monitoring is the ability to be fault tolerant. According to this work, they state that some systems omit mechanisms to ensure that the network agrees in one result based on the fact that the probability of occurrence of a bad result is small. However, this is a wrong assumption and might incur into an incomplete mechanism to deal with fault tolerant nodes. This derives the need to accommodate a consensus mechanism to provide a Byzantine or asymmetric fault tolerance.

4.1.2 Consensus Algorithms in Distributed Systems

The consensus algorithm is derived from the Byzantine generals problem proposed by Lamport et al. (Lamport, Shostak, & Pease, 1982). They state that a reliable computer system must be able to work upon the existence of malfunctioning parts which can be generating conflicting information. Therefore they propose an abstraction, where the Byzantine army with a group of generals in different locations has to agree in when to attack. The problem is that they can only rely on messengers and therefore they must consider the probability of the messenger being replaced or being a traitor. Then, they propose an algorithm that in this problem helps the generals to reach an agreement independently of the faults in the communication. From this, we can say that a consensus algorithm is a mechanism to allow different nodes in a network to agree on some information and work as a coherent group despite the failures of some of the nodes. From this Byzantine problem, there has been multiple studies including surveys (Fischer, 1983; Qin, Ma, Shi, & Wang, 2017), proposals like Paxos (Lamport et al., 2001) and Raft (Ongaro & Ousterhout, 2014). The study of Fischer et al. (Fischer, 1983), states that a Byzantine failure can be a faulty process that can send messages when it is not supposed to, send conflicting information, oppose the general view of the system, along with others. All consensus protocols have a number of t failures that can handle. Therefore it is said that a protocol is t-Byzantine resilient based on the failures it can handle. One of the most known consensus protocols is Paxos (Lamport et al., 2001). Although this protocol is considerably complicated and difficult to integrate, it became the foundation for consensus algorithms. These algorithms provide an intermediary layer for large-scale systems to agree upon specific information. Although the complications stated by Paxos, it was still implemented by multiple big tech companies in services like Amazon S3, Amazon DynamoDB, and Apache ZooKeeper. There are other companies like Google that developed their consensus algorithm (Burrows, 2006). In the recent years, a novel approach was proposed based on the limitations of Paxos, this proposal is Raft (Ongaro & Ousterhout, 2014). Companies like Hashicorp, support this new approach and it has gained enough popularity to have implementations in most of the programming languages like C, Java, GoLang, Python and more. The Raft protocol is considered to be better than Paxos since it decouples the leader election, log replication and safety, thus making it more modular. Moreover, it reduces the degree of non-determinism among servers of Paxos by defining clear and consistent rules for the communication between servers.

Recently, there has been another field where consensus algorithms have gained a lot of traction, which is in the field of cryptocurrencies. There are multiple algorithms as proof of work (PoW), where there is a brute force calculation of all the nodes to comply with a crypto-puzzle. There is

also proof of stake (PoS) where the consensus is done by the nodes that have more at stake in the system. Also, proof of elapsed time (PoET), where the consensus is done similarly to Raft, by waiting a random time, but this is done by the chipset instead of a software function. This last algorithm is backed and proposed by Intel. These algorithms help the cryptocurrencies to achieve a consensus, primarily focused on preventing intentionally users of tempering with the existing data. Given that cryptocurrencies represent a digital value, a malicius node would want to lie about the amount of currency it holds.

In the context of MANETs, the importance of a consensus algorithm is to achieve a global view by doing the computation in a distributed approach, therefore minimizing the risk of non-available nodes. It also is aimed at overcoming the faults induced by the mobility of the nodes. As mentioned before, the mobility of the nodes can induce some failures like off-range communications, fragmentation and therefore incomplete information. Relying on a consensus algorithm, it aides to enhance the robustness of the approaches by providing a distributed method to withstand these problems and to achieve a truly distributed agreement on a monitored value. This inherently will increase the accuracy and the stability of the approach.

4.1.3 Blockchain Technology

In the recent years, due to the economic crisis of 2008, there was a proposal of a digital currency called Bitcoin, proposed by the anonymous entity of Satoshi Nakamoto (Nakamoto, 2008). This proposal, elaborated on the idea of a currency which was entirely digital, it was not backed by any physical value as opposed by the euro or the dollar, the offer and demand determines its value, and finally, it is supported by cryptography, consensus algorithm, and a protocol. After this, the currency started to be known as a cryptocurrency, and a year later after its proposal, its author released the implementation of it. From this, there was a definite combination of technologies to make it possible that would later be known as the blockchain. The blockchain is a distributed ledger supported by cryptography functions to ensure the integrity of the data, a consensus algorithm to provide trust in a trust-less environment and a protocol to provide the network with the mechanisms to transmit the data. The protocol defines some basic packets that allow the network to work, which are:

a) transactions, which contain the information of a money transaction between two entities

b) and a block, which is created by a special type of node (or role) and contains a set of validated transactions.

A network running nodes using blockchain is known as a blockchain network. Each node or participant of the network can be one or more of three roles:

a) users (generate transactions)

b) nodes (check the consistency of validated blocks)

c) and miners (agree and validate transactions on a block).

The block is the center packet of the blockchain, and its most important value is its id. The body of a block is a list of transactions (A is transferring X amount to B, and so on). The header of a block is a collection of four pieces of metadata:

a) the id of the previous block

b) a nonce generated by the consensus algorithm run by the miners

c) a timestamp with the time of the creation

d) and a Merkle Tree Root of the transactions included in the block.

This four pieces of metadata are concatenated and double-hashed (SHA256) into the blocks id. Finally, the block is added to the chain. Where due to the cryptographic arrangements and integrated connections of the data, it is virtually impossible to modify something in a block. This concept has become widely popular not only in the economic context but also in the computer science context.

4.2 A Fully Distributed Monitoring Architecture

We propose a fully distributed architecture by relying on a distributed monitoring ledger and a consensus algorithm. Through this, we intend to provide a mechanism capable of monitoring a network with multiple probes and multiples points of observation. Our approach relies on a protocol that we define to support the multiple procedures of the network. The protocol depends on a packet that we have defined. Each packet has two parts, a header, and a body. We define four different types of packet: query, transaction, block, and consensus. As a preliminary definition, the query packet is generated by a node or external entity. This packet is broadcasted to the network. Then each node analyzes the query and produces a transaction. The transaction packet is the reply per node of the query, and as the name implies, it contains a transaction. A transaction is defined as the result of the verification of a local property, which is defined in the following sections. The query packet contains the information for the source of information, local projection criteria, and global validation constraints. It is only generated if the node is one of the sources of information and complies with the queried criteria. Therefore a query in the network can generate zero to N number of transactions, being N the number of nodes in the network. The consensus packet is generated to coordinate the election of the validator or consenser based on the consensus algorithm rules. Based on this consensus mechanism, the leader validates a global property based on all local properties generated by the query. This generates a result, which is assembled into a block. The block packet is the final result of a query, which contains a group of transactions and the results based on the query. To adequately define each part of our architecture, we must first define the messages, events, local properties and global properties. For this formalization, we get inspired by previous work performed in the context of services orchestration (Nguyen, Poizat, & Zaïdi, 2012) and in the context of SIP protocol (Che, Maag, Nguyen, & Zaïdi, 2015).

4.2.1 Distributed Monitoring Mathematical Definitions

Messages Under Observation

The network is interchanging information packages. This means that a node A is sending messages to node Z. The path between both nodes is changing due to mobility and availability in the MANET. Therefore, it means that a packet between these two nodes can take any finite amount of hops. Each node is capable of doing some actions based on the messages. From these, we define a set of basic actions that can be performed by every node for the communication of information.

Definition 4.1 (Actions)**.** An action is defined by the following grammar:

$$\Omega ::= New \mid Income \mid Outcome \mid Received$$

Where New refers to the creation of a new message, $Income$ refers to the reception of an incoming message, $Outcome$ to the sending of an outgoing message and $Received$ to the acknowledgement of a received message.

It is worth mentioning that any other grammar defining other actions for any other protocol is possible.

Example 1. *A node A, wants to send a message to node Z. Then it creates the message, therefore an action of New. Followed by an action of Outcome, to send it to the next hop of the routing mechanism.*

Example 2. *A node M in the middle of the routing between two nodes A and Z. This node receives a message from another node. Therefore this generates an action of Income. After some internal processing by the routing module, it concludes that the package is not for him and therefore it must forward it. Due to this, another action is generated, which would be an Outcome.*

Keep in mind that how the routing algorithm decides the next hop is out of the scope of this work. We assume that this is defined using the embedded routing protocol and each node will know what to do up to the reception of a packet.

Given the set of action names (Ω), we can notice that more information is needed to model an actual message. For example the destination of the message, next hop or previous hop, an even more advanced characteristics as status, checksum, protocol, etc. Therefore relying on the grammar for the actions, we can define a message.

Definition 4.2 (Message)**.** Given a finite set of action names Ω, of labels $\mathcal{L}$, and of atomic data values $\mathcal{D}$, a message m takes the following form:

$$m = o(l_1 = v_1, \ldots, l_n = v_n)$$

where $o \in \Omega$ represents the action. The composite data of the message is represented by a set $\{l_1 = v_1, \ldots, l_n = v_n\}$, rewritten as $(\bar{l} = \bar{v})$ for short, in which each field of this data structure is pointed by a label $l_i \in \mathcal{L}$ and its value is $v_i \in \mathcal{D}$.

In this work, we consider $\mathcal{L}$ as any field name of a message (*e.g.*, a routing packet with the fields destination, checksum, etc.) and $\mathcal{D} = integer \cup string$.

Example 3. *The first message from the Example 1 is formulated as the following:*

$m_1 =$ `New` *(source = "*`10.20.5.1`*", destination = "*`10.20.5.50`*",*
tag = 127, protocol = "`ospf`*", timestamp = 1520870382*
checksum = "`2cbcf04ddfce7058c8b4f41e60150fed`*")*

where "`10.20.5.1`*" is the IPv4 address of node A and "*`10.20.5.50`*" the IPv4 address of node Z. Then the checksum property is the result of the hash of the data of the message, for the purposes of this example this part is omitted and a random value is assumed. The timestamp is the Unix epoch time of the moment of the event. Other information can be modeled for a message as the protocol and tag.*

Example 4. *An Income and Outcome messages are formulated as the following:*

$m_2 =$ `Income` *(source = "*`10.20.5.1`*", destination = "*`10.20.5.50`*",*
previousHop = "`10.20.5.5`*", actualHop = "*`10.20.5.7`*",*
tag = 127, protocol = "`ospf`*", timestamp = 1520870383*
checksum = "`2cbcf04ddfce7058c8b4f41e60150fed`*")*

$m_3 =$ `Outcome` *(source = "*`10.20.5.1`*", destination = "*`10.20.5.50`*",*
nextHop = "`10.20.5.9`*", actualHop = "*`10.20.5.7`*",*
tag = 127, protocol = "`ospf`*", timestamp = 1520870383*
checksum = "`2cbcf04ddfce7058c8b4f41e60150fed`*")*

where "`10.20.5.7`*" is the IPv4 address of node M, "*`10.20.5.50`*" the IPv4 address of node Z and "*`10.20.5.5`*" and "*`10.20.5.9`*" are the two IPv4 addresses of two intermediary nodes between the package in transit from node A and node Z. The timestamp, protocol and tag properties are for exemplification purposes.*

From these definitions, we say that a message is an instance of an event. Meaning that a message is the single occurrence of an event. In our case, an event is a routing event, *e.g.*, the creation of a packet, the reception or sending of a packet, and so on. Due to our monitoring purposes, these events are under evaluation. We, therefore, define a candidate event, which is a pair constituted by an event e and a predicate ϕ. This represents a message or set of messages which are an instance of e that are satisfied by ϕ. To define a candidate event, we need to first define a predicate and a term.

Definition 4.3 (Predicate)**.** A predicate ϕ is defined wrt. the following:

$$\phi ::= \neg\phi \mid \phi_1 \wedge \phi_2 \mid t_1 = t_2 \mid t_1 > t_2$$

where t_1 and t_2 are terms, a term being either a constant or a variable, *e.g.*, $t ::= v \mid x$ where v is an atomic value and x is a variable.

Definition 4.4 (Predicate Dependency)**.** We tell that a predicate ϕ_j is dependent of a predicate ϕ_i, noted $\phi_j \sim \phi_i$, if ϕ_j is expressed using variables defined by ϕ_i.

Definition 4.5 (Candidate Event)**.** A candidate event (CE) is a pair $o(l_1 = x_1, \ldots, l_n = x_n)/\phi(x_1, \ldots, x_n)$, denoted by $o(\bar{x})/\phi(\bar{x})$, where $o(l_1 = x_1, \ldots, l_n = x_n)$ is a message and $\phi(x_1, \ldots, x_n)$ is a predicate. The predicate can be omitted if it is true.

Example 5. *To define a CE for a message based on the action* `New`*, we write:*

$$CE_1 = \texttt{New}\,(source = x)/(x = \text{“}\texttt{10.20.5.1}\text{”})$$

which represents any `New` *message whose source is “*`10.20.5.1`*”.*

Through these definitions we are able to formalize any routing message with their respective properties and more over to represent it using a candidate event for monitoring purposes. It is important to point out that a CE represents a set of messages. From these we can proceed to define local and global properties and further operations.

Local Property

Each node in a network participates in the network operations like routing packets. Not necessarily all nodes and not at all times, this depends on mobility and many other factors. Nonetheless, based on these interactions, each node will generate its own log of information. This information gives each node a unique local perspective of the network. This local perspective is defined through local properties. Therefore, the evaluation of a set of local properties derives a local view of the node. The evaluation mentioned above can derive an empty result, which means that one node cannot satisfy the given local properties. We define a local property $\mathcal{P}$, which is used to represent the behavior to be monitored in a node. In our specific case, these properties are validated against the given logs for the routing information of a node. The idea is that a local property can be evaluated using any data that can be represented as a finite stream of messages. The local property is expressed as a combination of a conditional statement and a control flow statement. This means that the conditional statement if a context is satisfied therefore a consequence must exist for the given context, *e.g.*, IF context THEN consequence. For example, let us look at Example 4, the logic is if a node in the middle of the routing of a packet destined to another node, it should receive an *Income* message and then it should forward an *Outcome* message. Moreover, for the control flow statement, it means that, given a set or sets of messages, it will iterate performing a predefined operation. The evaluation of the CEs and its result, a

set of messages, is explained in detail in the following sections. We proceed to formalize the local property as following:

Definition 4.6 (Local Property)**.** A local property $\mathcal{P}$ is a 3-tuple (cont, conseq, op) composed of a Context cont AND/OR a Consequence conseq AND/OR an Operation op. It is denoted as:

$$\mathcal{P} ::= cont \rightarrow conseq \times op$$

where:

- *Context* is a sequence of CEs, $\langle e_1/\phi_1, e_2/\phi_2, \ldots, e_n/\phi_n \rangle$ where it may exist pairs of indexes $i, j \in \{1, \ldots n\}, i < j$ such that $\phi_j \sim \phi_i$.
- *Consequence* is a set of CEs, *e.g.*, $\{CE_1, \ldots, CE_m\}$. Therefore, according to a Context, a Consequence can or cannot be validated. Note that a Consequence can be empty. In that case, the property just focuses on the Operation op.
- *Operation* is an operator that describes the operation followed by a term or terms, *e.g.*, $(exist \mid sum \mid avg \mid count \mid *)$.

  An Operation is a function that considers the terms of cont if conseq= $\emptyset$ or all terms in cont$\rightarrow$conseq if op$\neq \emptyset$.

  Note that op is not mandatory, it may be unnecessary for some properties. If op is omitted, the result of the validation of $\mathcal{P}$ is a boolean.

Example 6. *Let us look at the case where a node in the middle of the routing of a packet is destined to another node. If a node receives a message which is destined to another node, it should forward it to the next hop defined by the routing algorithm. Therefore for a node different from the source or the destination, it should be true that if there is an Income message to node Z, there should exist an Outcome message to node Z. This local property is defined as follows:*

$$\begin{aligned}\mathcal{P}_1 &= \langle \texttt{Income}(destination = x_1)/(x_1 == \text{“}\texttt{10.20.5.50}\text{”}) \rangle \\ &\quad \rightarrow \{\texttt{Outcome}(destination = x_2)/(x_2 == \text{“}\texttt{10.20.5.50}\text{”})\}\end{aligned}$$

In a more general case, such a local property could be that for all incoming messages, there exists an outcome message with the same recipient.

$$\begin{aligned}\mathcal{P}_2 &= \langle \texttt{Income}(destination = x_1) \rangle \\ &\quad \rightarrow \{\texttt{Outcome}(destination = x_2)/(x_1 == x_2)\}\end{aligned}$$

Note that, for these both cases, we do not have an operation, therefore it is assumed that the operation between the context and the consequence is of existence. Meaning that for every item of the context, it must exist an item in the consequence. Which also implies that the result is boolean.

Example 7. *Lets us look at the case where an operator is present. Looking at a simpler task, we want to count the number of messages that we have received from a given node. Therefore, we only need to project a CE with the given destination and then iterate over the results to count the elements of the projection. This local property is exemplified as follows:*

$$\mathcal{P}_3 = \langle \texttt{Income}(destination = x_1)/(x_1 == \text{“}10.20.5.50\text{”}) \rangle \times \{count\ x_1\}$$

Note that, for this specific case, we only need the context and the operation. Therefore, we omit the consequence and the operation acts over the results of the context. The final result domain depends on the operator, in this case, the result is an integer.

Example 8. *Let us look at a case where an operator is present as well as both context and consequence. Looking at a more complex task, we want to know the average time it takes for a node to process an Input and to produce the corresponding Output. First, lets assume that we are looking at the case where the message destination is given. Due to space limitations we abbreviate "destination" as "dest" and "timestamp" as "time". This local property is defined as follows:*

$$\begin{aligned}\mathcal{P}_4 = {} & \langle \texttt{Income}(dest = x_1, time = x_2))/(x_1 == \text{“}10.20.5.50\text{”}) \rangle \\ & \rightarrow \{\texttt{Outcome}(dest = x_3, time = x_4)/(x_3 == \text{“}10.20.5.50\text{”})\} \\ & \times \{average\ (x_4 - x_2)\}\end{aligned}$$

The final result domain depends on the operator, in this case, the result is a float or double.

Global Property

Given the distributed nature of our architecture, it is important to note that each node holds its own perspective from its set of information. However, some of the data hold locally from some nodes, intersects with other local data of another set of nodes. This intersection of local information gives a more broad view of the network. Therefore we say that there is a network perspective based on the aggregation of the local perspectives. This means that by relying on local perspectives, it can be determined a global view of a property of the network. A global view provides more general information since it aggregates over local views. For example, if we want to model if node A created a message to node B or if we want to model that node B received a message that was originated from node A, both of this cases can be modeled through a local property. However, if we want to ensure that the integrity of the message generated from node A to node B was maintained through the whole routing procedure, we need more information than just local views. For this specific example, we could verify that the data checksum generated by the source node A is the same as the data checksum by the receiving node Z. We are assuming that the field checksum of a packet is defined as the result of calculating a hash function of the body of the packet. This would give us a degree of certainty that the data generated was the data received but would not ensure that the integrity was maintained through the whole process. Therefore, we would need all the nodes involved in the communication between node A

to node Z and validate the checksum of each to verify that the integrity is maintained. This gives us a complete certainty of the integrity for a given packet, and maybe it is not necessary all nodes in the network, but for the effects of the example, we assume that all the nodes in the network are of interest. For this kind of scenarios, where the verification needs two or more point of observations, we need to define a global property. The global property is also expressed as a conditional statement followed by a control flow statement. This means that if a context is satisfied therefore a consequence must exist for the given context. This property is defined on top of local properties. Therefore we define a global property as follows:

Definition 4.7 (Global Property)**.** A global property **G** is a 3-tuple (SET, SET', op) denoted as:

$$\mathbf{G} ::= SET \Rightarrow SET' \times op$$

where SET and SET' are two sets of local properties and op is an Operation as defined for local properties.

Example 9. *Let us look at the case where we want to check that the integrity from the source node is indeed the same of the receiving node. For the ease of the understanding of this example, we assume that we are looking at only one message in particular going from one node to another identified with an id equal to "1234". And we assume that the field checksum was properly calculated at the time of the transmission. Therefore, we rely on two local properties. The first is defined as the creation and sending of the message from one node. And the second is defined as the receiving and acknowledgement of the message from another node.*

$$\begin{aligned}
\mathcal{P}_1 &= \langle \texttt{New}(id = x_1)/(x_1 == 1234)\rangle \\
&\quad \rightarrow \{\texttt{Outcome}(id = x_2)/(x_2 == 1234)\} \\
\mathcal{P}_2 &= \langle \texttt{Income}(id = x_1)/(x_1 == 1234)\rangle \\
&\quad \rightarrow \{\texttt{Received}(id = x_2)/(x_2 == 1234)\} \\
\boldsymbol{G}_1 &= \{\, \mathcal{P}_1, \mathcal{P}_2 \,\} \\
&\quad \Rightarrow \{\, \langle \texttt{New}(checksum = y_1)\rangle \\
&\qquad \rightarrow \{\texttt{Received}(checksum = y_2)/(y_1 == y_2)\}\}
\end{aligned}$$

Example 10. *Let us look at the case where we want to check that the integrity from the source node is indeed the same of the receiving node. We assume that the field checksum was properly calculated at the time of the transmission. We check that the checksum is maintained through the whole communication. Therefore, we rely on the three following local properties.*

$$\begin{aligned}
\mathcal{P}_1 &= \langle \texttt{New}(id = x_1)/(x_1 == 1)\rangle \rightarrow \{\texttt{Outcome}(id = x_2)/(x_2 == 1)\} \\
\mathcal{P}_2 &= \langle \texttt{Income}(id = x_1)/(x_1 == 1)\rangle \rightarrow \{\texttt{Outcome}(id = x_2)/(x_2 == 1)\} \\
\mathcal{P}_3 &= \langle \texttt{Income}(id = x_1)/(x_1 == 1)\rangle \rightarrow \{\texttt{Received}(id = x_2)/(x_2 == 1)\} \\
\boldsymbol{G}_1 &= \{\, \mathcal{P}_1, \mathcal{P}_2, \mathcal{P}_3 \,\} \Rightarrow \{ \\
&\quad \langle \texttt{New}(checksum = y_1)\rangle \rightarrow \{\texttt{Received}(checksum = y_2)/(y_1 == y_2)\}, \\
&\quad \langle \texttt{Income}(checksum = y_3)\rangle \rightarrow \{\texttt{Outcome}(checksum = y_4)/(y_3 == y_4)\}\}
\end{aligned}$$

Example 11. *Let us look at a case where an operator is present. Lets say, that we want to know the average processing time between the Income and the Outcome of a specific message in the network. Therefore, we need to rely on the time the Income message was registered and the time the Outcome message was registered by the node. For this we rely on a timestamp label, which tell us the creation time of the packet. The local property would look like Example 8. In the scope of a global property, we would modify our local property, and move the operator to the global scope. Therefore, it would look like the following:*

$$\begin{aligned}\mathcal{P}_1 &= \langle \texttt{Income}(id = x_1, time = x_2))/(x_1 == 1234)\rangle \\ &\quad \rightarrow \{\texttt{Outcome}(id = x_3, time = x_4)/(x_3 == 1234)\}\end{aligned}$$

$$\boldsymbol{G}_1 = \{\, \mathcal{P}_1 \,\} \times \{average\ (x_4 - x_2)\}$$

Example 12. *Let us look at a more complex case where an operator is present. Since we work in the context of MANETs, we might need to measure the average transmission time of the messages sent. For this, we would need to measure the delta of time between the time when a message was sent and the time when the message was received by the next hop. Therefore, this metric cannot be calculated by single local view, but it can only been measured by the global view. For the purposes of this example, we assume that the packet contains the address of the actual hop with the label "hop" and of the previous hop with the label "prevHop". Therefore, it is defined as:*

$$\begin{aligned}\mathcal{P}_1 &= \langle \texttt{Outcome}(id = x_1)/(x_1 == 1234)\rangle \\ \mathcal{P}_2 &= \langle \texttt{Income}(id = x_2)/(x_2 == 1234)\rangle \\ \boldsymbol{G}_1 &= \{\, \mathcal{P}_1,\ \mathcal{P}_2 \,\} \Rightarrow \{ \\ &\qquad \langle \texttt{Income}(prevHop = y_1, timestamp = y_2)\rangle \\ &\qquad \rightarrow \{\texttt{Outcome}(hop = y_3, timestamp = y_4)/(y_1 == y_3)\}\} \\ &\quad \times \{average\ (y_4 - y_2)\}\end{aligned}$$

4.2.2 Network Protocol Internals: Procedures

Query procedure

Our approach is reactive to a query generated by a node in the network or by an external request. The query is assumed to request information for monitoring objectives from all the nodes in the network. Therefore the query is generated and then all the network computes a result. The query is broadcasted in a best effort approach to all nodes in the network. When each node receives a query, it evaluates it based on the criteria given to get a result. If the node has a result, it then generates a transaction packet and it broadcasts it to the network. The transaction packet is explained in detail in Section 4.2.2. The result generated by a node is considered a partial response to the query given that it only takes into account its local view. The query can rely on local properties and global properties, not necessarily both. The query packet has a body formalized as follows:

Definition 4.8 (Query)**.** The body of the query packet is defined by the following tuple:

$$Query = \{ G_{PROP}, T_o, T_r \}$$

where G_{PROP} is a global property, T_o is a timeout defined in milliseconds by an integer and T_r is an optional number of transactions defined as any number greater than zero as an integer.

The idea is that a query is transmitted to the network. Then evaluated by each node to create local views and later to calculate the global view and final result of the query. The evaluation of the query by a single node, can derive a local view which is encapsulated by our approach as a transaction. We are assuming that each query contains only one global property. If there is a need for more global properties, each one of them should be executed using different queries. The query integrates one required and one optional parameter, which are a timeout and a number of transactions. Every query operates over a MANET, then it has to overcome availability, mobility, fragmentation and other inherent properties of these networks. Therefore, we provide some parameters to ensure a deterministic approach of completing a query. The timeout, as the name implies, is just a time to wait and afterwards all the existing transactions are aggregated. The number of transactions, if present, is a limit on the quantity of existing transactions for the given query, therefore until this threshold is met the query is not complete. If both parameters are present, then the query will be completed when one of them is satisfied.

The Algorithm 2 describes the broadcast of a packet. Basically, it sends a packet to the network if it has not been sent before. Finally, the Algorithm 4 describes the behaviour to analyze the timeout parameter of the query. In essence, it waits for the time defined in the query (Line 2). When the time is reached it checks that the query has no result (Line 3), which is determined by the existence of a corresponding block for the query (explained in detail in Section 4.2.2). If there is no result for the query, then it triggers the consensus (Line 4) to complete it and therefore generate a result for the query.

The behavior of a node upon the reception of a query is depicted in Algorithm 3. The first step is to trigger the procedure that waits for the timeout of the query (Line 3), which is run asynchronously or in parallel. Then, it iterates over the local properties of a query (Line 4). For each one of them, it first evaluates the local property and the set of the node local traces (Line 5). The description of the function that evaluates the local property can be found in Section 4.2.2 in Algorithm 6. If the result is not empty (Line 6) then it creates and sends the transaction (Line 7).

Since this procedure relies on non-recursive nor non-loop statements, we can say that the complexity of this procedure is $O(1)$. The Algorithm 3, relies on Algorithm 6 whose complexity is $O(n^2 + n)$ (explained in detail in Section 4.2.2). Therefore, we can say that the complexity of Algorithm 3 is $O(k * (n^2 + n))$. Where k is the number of local properties of the query.

Example 13. *Let us look at a query that requests the integrity of messages. Lets take as a reference the Example 10. If we assume that we are looking at the already existing communications, and therefore we assume that the processing time for this is not considerable. Then the example looks like this:*

Algorithm 2 Broadcast packet to the network

```
1: broadcastPackets ← array()
2: procedure BROADCAST(Packet)
3:     if Packet is not in broadcastPackets then
4:         sendBROADCAST(Packet)
5:         broadcastPackets[] ← Packet
6:     end if
7: end procedure
```

Algorithm 3 Reception of a Query in a node

```
1: procedure RECEIVEQUERY(Q)
2:     BROADCAST(Q)
3:     WAITTIMEOUT(Q)                          ▷ This is run asynchronously
4:     for each Q.LPs as LP do
5:         result ← EVALLP(LP, myTraces)
6:         if result is not empty then
7:             SENDTRANSACTION(LP, result, myTraces′)
8:         end if
9:     end for
10: end procedure
```

Algorithm 4 Check query timeout

```
1: procedure WAITTIMEOUT(Q)
2:     WAIT(Q.T_o)
3:     if Q has no result then
4:         TRIGGERCONSENSUS
5:     end if
6: end procedure
```

$$Q_1 = \{ G_1, 2000 \}$$

This means that once the query is disseminated to the network, there is a limited time of 2000ms to complete the query, and after that time, the query is completed with the existing information collected in the time window given. But lets say that we need at least 5 points of observation, therefore we could add a number of transactions. Then the example looks like this:

$$Q_1 = \{ G_1, 2000, 5 \}$$

Transaction procedure

Upon the reception of the query, the nodes need to parse it and evaluate it. Relying on our previous definitions, if the query is composed by a global property, this means that it contains a set of local properties. As we mentioned before, the global property is only evaluated when there exists a result of multiple local properties. Therefore, a transaction corresponds to the result of evaluating a local property. From this evaluation, a result is derived, which is added to a transaction packet and broadcasted to the network. If the result is different than null, a node can generate only one transaction per query. The contrary case when the result is null, meaning that the node does not contain any information that can suffice the local property, then there is no transaction generated by the node. So before we define the actual transaction packet, we need to define the evaluation of a local property. A global property contains one or more local properties. Each local property contains one or more candidate events. Therefore, to define the evaluation of the query, we need to evaluate all local properties and by transitivity all candidate events of them. However, to evaluate the smallest of the entities, we need a set of data to rely on. For this, we define the data source as a trace segment, which is defined as $\gamma = \langle m_1, \ldots, m_k \rangle$, a list of messages. The traces are instantiated traces coming from the logs of the nodes. This is explained in detail in Section 4.3.1. We first define the evaluation of a candidate event with Algorithm 5.

This algorithm defines two sections, a projection and a selection. The first section (Line 2) defines a projection of a candidate event (CE) from the trace segment. This operation projects all traces that contain the corresponding message and field names required by it. We check that the message name is the same and that the list of field names and values corresponds as well. Then, if a message contains the required criteria, it is considered as part of the projection and is added to an array.

The second section defines a selection from the projected traces. To define this selection (Line 3), the predicate of the CE is evaluated. By definition of a predicate, we only have boolean operators, the result is boolean as well ($true, false$). This means that the operation selects the messages that comply with the predicate. Finally, the result from both sections is returned. Let us note that the result contains a subset of the original trace segment that complies with one candidate event. The number of its elements can be zero up to the number of elements of the original trace segment. This will depend on the specification of the candidate event.

Each section of this algorithm can be considered as a loop. Therefore we can say that the projection (Line 2) has a complexity of $O(n)$ and the selection (Line 3) a complexity of $O(m)$. Where n is the number of elements in the set of $Traces$ and m is the number of elements of the projection, where $m \leq n$. In the worst case scenario, we can assume that $m == n$ and therefore the complexity of Algorithm 5 is $O(n)$.

Algorithm 5 Evaluation of a CE in a list of traces

Precondition: Π refers to the projection operation and σ refers to the selection operation from relation algebra.

1: **procedure** EVALCE($CE, Traces$)
2: $projection \leftarrow \Pi_{CE.o,CE.l_i}(Traces)$
3: $selection \leftarrow \sigma_{\phi(v,x)}(projection)$
4: **return** $selection$
5: **end procedure**

From this, we can define the evaluation of local property by considering an algorithm that analyzes the context, the consequence, the operation and then their results in a conditional statement. For that, we define the Algorithm 6. The algorithm has three sections. The first one (Line 2) is where a subset of traces are derived from the context. The second part (Line 6) is where a subset of traces are derived for the consequence. Finally, we evaluate the operation of the local property (Line 10). First, we evaluate if an operation exists, then we apply the given operation to the list of messages. Otherwise, we fall back to the default behavior (Line 13) and return the result of the conjunction that for every context it should exist a consequence.

The complexity of this algorithm can be broken down into the three parts. The first part (Line 2) and the second part (Line 6) are loops that calls $evalCE$ which has complexity $O(n)$. Therefore we can say that each part has a complexity of $O(m * n)$, where m is the number of candidate events in the context and consequence correspondently. We can assume that in the worst case the number of candidate events is the same for the context and consequence, therefore for both loops, the complexity is $O(m * n)$. For simplicity purposes and assuming the worst case scenario of the nested loops we could say that $m == n$, and therefore the complexity is $O(n^2)$. Finally, the last part (Line 10 and Line 13) can be considered a loop i.e. $O(n)$. Then the complexity of Algorithm 6 is $O(n^2 + n)$.

Algorithm 6 Evaluation of a Local Property in a list of traces

1: **procedure** EVALLP($LP, Traces$)
2: $evalcontext \leftarrow$ map()
3: **for each** $LP.Context.CEs$ as CE **do**
4: $evalcontext[] \leftarrow$ EVALCE($CE, Traces$)
5: **end for**
6: $evalconsequence \leftarrow$ map()
7: **for each** $LP.Consequence.CEs$ as CE **do**
8: $evalconsequence[] \leftarrow$ EVALCE($CE, Traces$)
9: **end for**
10: **if** $LP.Operation$ is not empty **then**
11: **return** $LP.Operation^{n}_{i=1}(evalcontext[i],\ evalconsequence[i])$
12: **end if**
13: **return** $\bigwedge_{i=1}^{n} evalcontext[i] \exists evalconsequence[i]$
14: **end procedure**

With these two algorithms, we can define the transaction packet as the tuple defined as a property,

the evaluation of that property and the traces that satisfy the local property.

Definition 4.9 (Transaction)**.** The body of the transaction packet is defined as follows:

$$Transaction = \{\ P, evalLP(P, \Gamma_{local}),\ \Gamma'_{local}\ \}$$

where P is a local property, $evalLP(P, \Gamma_{local})$ refers to the result of the evaluation of P through the set of traces Γ_{local}, and $\Gamma'_{local} \subset \Gamma_{local}$ represents the traces that were effective for the evaluation of P (i.e., the above-mentioned subset, indeed, not all traces from the list are locally usable or useful for evaluating the formula, this can depends on the node processing the property evaluation). Note that $evalLP(P, \Gamma_{local})$ can be either boolean, integer, double or any other data type defined by the Operation of the local property.

It is important to mention that, if the evaluation of the local property returns false, then no transaction packet is generated. But if a packet is generated, then this packet is broadcasted to the network. The behavior of a node upon the reception of a transaction is depicted in Algorithm 7. This algorithm starts by broadcasting the transaction to ensure the transmission of it (Line 2). Then it stores the transaction (Line 3). Followed by getting the corresponding query (Line 4). Finally it checks if there is enough transactions based on the parameters of the query (Line 5) and if this is satisfied, then it triggers the consensus procedure (Line 6). The consensus procedure is explained in detail in Section 4.2.2. This algorithm does not contain loops nor recursive statements, therefore we can say that the complexity is $O(1)$.

Algorithm 7 Reception of a Transaction by a node

```
1: procedure RECEIVETRANSACTION(T)
2:     BROADCAST(T)
3:     transactions[] ← T
4:     Q ← get query for T
5:     if Q.T_r exists and is >= than Ts of Q then
6:         TRIGGERCONSENSUS
7:     end if
8: end procedure
```

Block Procedure

All the nodes in the network contain the queries and the transactions generated by the queries. Based on the query restrictions and the transactions, every node is capable of deciding when to aggregate the data. This is referred in Algorithm 7 and Algorithm 3. Therefore, when these criteria are met, all nodes employ a consensus algorithm to dictate a node to perform this task. Although the consensus algorithm is explained in detail in the following section, note that this is a separate procedure and that the result is an elected leader node. The temporal leader node takes all the transactions it has from a given query, aggregates them and analyzes the global property. To determine the evaluation of a global property, we rely on the definition of the evaluation of the local property. The Algorithm 8 defines the evaluation of a global property. It is a recursion over the same concept on how to evaluate a local

property. It has three parts, the first getting the traces of the context, then the traces of the consequence and finally returning the result of the operation. This procedure yields a result which is consolidated and sent as a block packet.

The complexity of this algorithm relies on the Algorithm 6, whose complexity is $O(n^2 + n)$. For the first two parts, we can say that the complexity is $O(j * (n^2 + n))$, where j is the number of local properties of the context and consequence correspondently. The last part can be considered a loop, i.e., $O(n)$. Finally, the complexity of Algorithm 8 can be considered $O(j * (n^2 + n) + n)$.

Algorithm 8 Evaluation of a Global Property in a list of traces

```
1:  procedure evalGlobalProperty(GP, Traces)
2:      context ← map()
3:      for each GP.Context.LPs as LP do
4:          context[] ← traces of evalLP(LP, Traces)
5:      end for
6:      consequence ← map()
7:      for each GP.Consequence.LPs as LP do
8:          consequence[] ← traces of evalLP(LP, Traces)
9:      end for
10:     if GP.Operation is not empty then
11:         return GP.Operation_{i=1}^{n}(context[i], consequence[i])
12:     end if
13:     return ⋀_{i=1}^{n} context[i] ∃ consequence[i]
14: end procedure
```

Definition 4.10 (Block)**.** The body of the block packet is defined as follows:

$$Block ::= \{\ G,\ evalGP(G, \gamma_{local_i}),\ \gamma_{local_i}\ \}$$

where G is a global property, γ_{local_i} is a set of traces from the local properties of G and $evalGP(G, \gamma_{local_i})$ refers to the result of evaluation the global property in all the local trace segments.

The behavior of a node upon the reception of the consensus end is depicted in Algorithm 10. As mentioned before, once the consensus algorithm elects a leader, the aggregation procedure is done. This procedure consolidates the query and the transactions of the query (Line 2-Line 3). Then the global property is evaluated as explained before (Line 4). Then based on the Merkle tree of the transactions, timestamp, previous block id and a generated random string, the block id is generated to ensure integrity. Finally, the block is consolidated, then broadcasted to the network (Line 5) and added to the ledger (Line 6).

The behavior of a node upon the reception of a block is depicted in Algorithm 9. The other nodes, upon the reception of the block, verify the block packet to make sure that the information is congruent. The verification is done by evaluating the query with the data that the node has. This should have the same result as the one transmitted in the block (Line 3). If the block passes this second verification (Line 4), then it is added to the nodes ledger (Line 5) and then broadcasted to the network. Otherwise, the block is rejected by the node and announced to the network (Line 8). Given that both procedures do not contain recursion nor loop statements, the complexity for both can be considered $O(1)$.

With these procedures, we can define the block packet, as the tuple with a global property, the evaluation of the global property and the traces that satisfy the global property. It is important to mention that, given the inherent properties of the MANET, there can be incongruent scenarios for some nodes. Let us look at the case of network fragmentation, so all nodes have the same information, but one of the nodes in the network went off range for a period of time, and it has one or more missing transactions. This means that when this node tries to verify the block, it will fail to do so. In this case, each node has to keep track of what the other nodes are broadcasting, and if the majority of the network approves a block that was rejected, then the node should accept it. Another case along the same line is if a node verified a block, but it hears an error for the same block from another node. Once again, the majority of the network is what defines the outcome. It is worth mentioning that, if a node verifies a block, but it receives that the majority of the network has acknowledged the block as an error, then the block should be discarded. This case could happen due to network fragmentation, but at the time that a node sent its transaction to the network, it was off the range. Therefore, most nodes had a missing part. For simplicity purposes, each node agrees with the result on which the majority of the network agrees. For example, there is a network of 20 nodes. Then one node does not agrees on a specific block but he receives that the other 19 nodes agreed on it. In such case, he agrees with the network on the validity of the block.

Algorithm 9 Reception of a Block in a node

```
1:  procedure receiveBlock(B)
2:      T ← get Transactions for B.Q
3:      testResult ← evalGlobalProp(B.Q, ⋃T.traces)
4:      if testResult == B.result then
5:          addToChain(B)
6:          broadcast(B)
7:      else
8:          sendError
9:      end if
10: end procedure
```

Algorithm 10 Reception of the consensus end

```
1: procedure receiveConsensusEnd
2:     Q ← get Query
3:     T ← get Transactions for Q
4:     globalResult ← evalGlobalProp(Q, ⋃T.traces)
5:     sendBlock(Q, traces′, globalResult)
6:     addToChain(newBlock)
7: end procedure
```

Consensus Procedure

Our approach aims to get the responses of the nodes to the query, but taking advantage of the distributed nature of the approach, also to compute a global result based on these responses to provide a more robust conclusion. We propose to utilize a consensus algorithm as a mechanism for grouping the transactions, validating them on a higher level and finally providing the complete result of the query to the network (Figure 4.1). Every time a response to a query is generated by a node to the network, transaction, the nodes verify if the parameters (timeout and/or the number of transactions) of the query

completion are met. If the conditions are met, then the consensus algorithm is triggered to elect a validator. Therefore this entity groups the transactions of the query and then it validates the global property. Finally, a block is assembled with the group of transactions and the results of the validation, and it is broadcasted to the network. After this, the elected leader goes back to a sleep state, and all the nodes stop any consensus activity until it is again needed. When it is needed again, a new election is held, and a new leader is elected. This is an essential factor of our approach. Given the properties of the MANET, it is crucial that, at each round, any node can act as the validator. This makes the approach more dynamic, suitable for change and scalable. From the data perspective, it also makes it more democratized, fair and distributed, since all nodes can and would participate in the validation of the monitoring information.

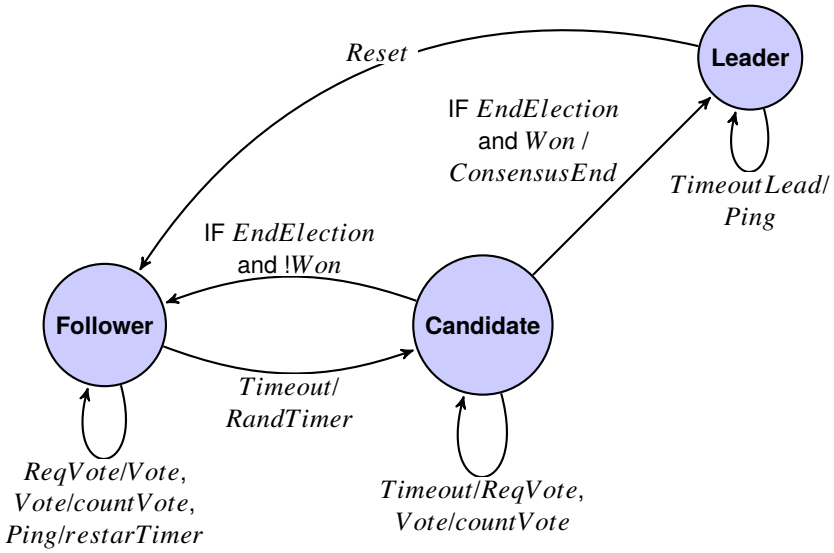


Figure 4.1 – IO FSM definition of the raft-inspired leader election

Relying on the definitions made by Raft (Ongaro & Ousterhout, 2014), we define our leader election algorithm illustrated as an input-output finite state machine (Figure 4.1). After the election is held, the leader evaluates the global property based on the transactions containing the local views. This generates a block packet that is broadcasted to the network, as explained in the previous section. Finally, the consensus is completed when the receiving nodes of the block verify it.

Any node participating in the election can be in three different states: Follower, Candidate or Leader. The Follower state is the most common state, and every node starts in this state. The idea is that if a node joins a network with an existing leader, they follow the leader. However, if a timeout is reached and it does not hear from any leader, then they switch to Candidate state. Aside from that, if they are requested to vote for a leader, they do so one time, and they vote for the one that they were requested first, therefore based by time and not by any other preference. Otherwise, all nodes vote, and at the same time they participate as watchers in the election. Therefore they keep track of the votes they hear aside from theirs. This gives the ability to the network to be sure that the leader is effectively the elected leader. If a malicious author decides to declare himself as the leader, the other nodes will know it, and they are able to ban this node. The Candidate state is for a postulating node to be a leader. Therefore the first thing they do is to do a random timeout and when that timeout is reached they request for votes. If they hear a vote either for them or for another participating node, they count it

nonetheless. If they receive a request for vote, they will fall back to the Follower state and vote for that candidate. It is important to note that, if two or more candidates clash with the same random time, and they request votes at the same time, they all return to the Follower state and after reaching a timeout a new election is held. By the end of the election, defined as a particular timeout, the node with the majority of nodes is the winner. Then that node announces itself as the leader, switches into the Leader state, and it closes the election. From this moment, the leader pings periodically the followers so they know of its existence. If the leader goes out of range or goes offline, another election will be held. Once a leader is elected, this node triggers the end of the consensus, giving place to the creation of the block and the completion of a query. It is worth mentioning that the leader will be pinging the followers as long as it is needed. However, once the block is created and transmitted it resets everything and remains in a sleep state until it is needed again. This applies to any node in the network, and the reset occurs either when the block is sent or received, depending on the node. Therefore, if there is no need for a leader, there is also no need for the exchange of messages of the given election leader protocol.

Clean and Restart Procedures

To avoid resource consumption, specifically storage space for each node, there is a procedure to free resources from past queries. This is a restart procedure, and this takes care of cleaning storage of all the blocks from a specific query, specific set of queries or all queries. Therefore, a query or multiple queries can be performed at any given point in time. Once there is enough information, this is defined by the node or external entity generating the query, it can be asked to the network to discard all blocks from these queries. Our approach intends that multiple queries can be performed in a defined and finite amount of time and that all these results are stored for enhancing the overall result. However, in the long term this can become a storage problem. Therefore the idea is to avoid these problems with this procedure.

4.3 Experiments for the Distributed Monitoring Aproach

We evaluate our proposal using an emulator built in-house described in Chapter 5. The testbed consisted in an implementation separated in four repositories of our proposal in the language Go (v1.9). Each repository represents a module of the complete implementation, it is separated into ledger[1], miner[2], router[3] and raft[4]. The implementation is separated into different modules to isolate the different procedures and to allow parallel execution of all the tasks without any problem. The ledger module refers to the entity in charge of storing the chain of blocks, generating the transactions and validating blocks. The miner module refers to the entity of starting the election process, and upon the reception of the results of the election, it creates the block for a given query. The router module refers to the entity that handles the packets from the network and sent it to the corresponding module, also broadcasts all packets and keeps track of it to avoid cyclic broadcast of the same packet. Finally, the raft module refers

[1]https://github.com/chepeftw/LedgerMANET
[2]https://github.com/chepeftw/MinerMANET
[3]https://github.com/chepeftw/RouterLedgerMANET
[4]https://github.com/chepeftw/raft

to the entity in charge of electing a leader. The idea is to identify primarily the completion time per query, the accuracy of the approach, the traffic it generates and how it compares to other approaches.

The completion time per query is defined as the time it takes from the generation of the query to the official result of the query represented as a block. This includes the time it takes to propagate the query, analyze it per node, generating the transactions, the consensus algorithm, the block creation and finally the broadcast of the query. The accuracy of our approach is defined by the validity of the blocks generated from the consensus algorithm. This means that every time a node receives a block, it validates the information of it and determines if the block is valid or not. Then this information is used to determine that for each query, the overall process is valid or not, and how many nodes agreed on this. It is essential to measure this since the MANETs are prone to fragmentation due to mobility. Therefore, it exists a need to determine the accuracy of the approach despite fragmentation or other MANET problematics. The traffic generated by our approach tells us how many packets are being generated and transmitted over the network. Ideally, it should transmit few packages for resource efficiency, but problems like fragmentation tend to make this difficult. Finally, we compare our approach to other approaches in the literature to determine the pros and cons of the approach at hand.

4.3.1 Implementation considerations

All the modules of the implementation depend on a packet definition and the logs that are being monitored.

Packet Structure Definition

The packet is defined in a JSON (Bray, 2014) format summarized in Figure 4.2 and contains multiple subparts to work with the overall implementation. The handling of the packet is done by the Type property in the structure. The different types that can exist are: a) query b) transaction c) block d) election e) raft_ReqVote f) raft_Vote g) raft_Ping h) raft_ConsensusEnd and i) raft_Reset. For example, if the packet is a query type, then the implementation handles it as a query and looks for the query structure inside the packet. For this specific type, the query structure depends on other structures (global property, local property, and candidate event) hence the arrows in Figure 4.2. This also means that it omits the transaction, block, and raft structure since they are not relevant to the query. Another example is for any of the raft types, the implementation looks for the raft structure in the packet and omits the query, transaction, and block.

Log Dependency Definition

The monitoring of our properties and our approach is based on analyzing a set of traces. The set of traces we relied on is the logs of the node. For this, we assumed that the log contains a predefined format. The format contains default information as timestamp and severity, but it also contains the action and labels.

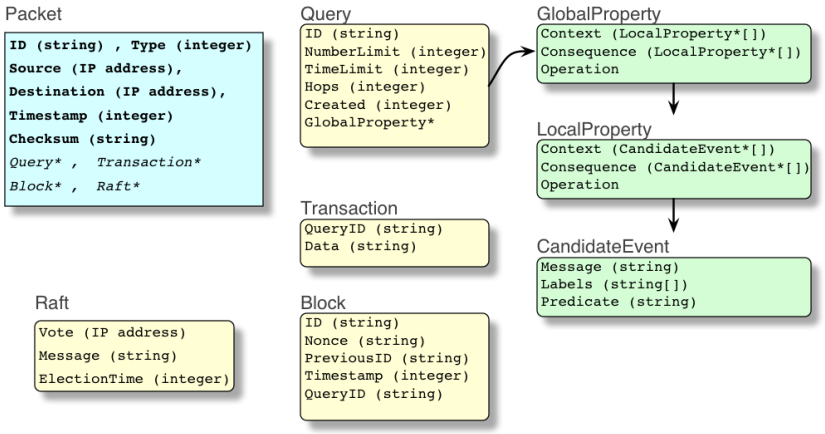


Figure 4.2 – Packet structure for the experiments for the distributed approach

Listing 4.1 – Log Format

```
timestamp other Action [label1=value1, ..., labelN=valueN]
```

For Example 3, assuming a random timestamp and severity of “info”, the log line is:

Listing 4.2 – Log line for Example 3

```
123456789 New [source="10.20.5.1", destination="10.20.5.50", tag=127, timestamp=1520870382, checksum="2
    cbcf04ddfce7058c8b4f41e60150fed", protocol="ospf" ]
```

For our implementation, we developed a small tool that parses the information from a tcpdump file and outputs our log format. The captured information is from a MANET where a node routes a packet to another node using OSPF. From this, we captured the information for all participating nodes and created the log we use. It can be seen as an extra step to achieve the log, but this modularization allows a more easy adaptation for other tools.

4.3.2 Experiments Setup

Test Case

We propose a test case where there is a MANET in a defined space with a fixed number of nodes. The nodes use the MAC protocol of 802.11a. The transport protocol is UDP with a data rate of 54Mbps and a node range of ≈125m. Each node starts in a random position and starts moving in a random direction at a fixed velocity. After the test case has started, there is an initial configuration time, equal to the number of nodes to randomize the position of the nodes. The idea is for the nodes, to shuffle from their original location. All of this is done to enhance the trustability of the results by using random values to ensure that the algorithm works in any given scenario and that any emulation parameter does not tie it. Then a node from the network, chosen randomly, triggers a query and is sent to the network. The query tries to determine the integrity of packet transmission with a $T_o = 60000$ and $T_r = 4$. Meaning that the global property of the query looks like Example 10, with a $T_o = 60000$ and $T_r = 4$, both values were chosen arbitrarily. Then the nodes that can answer this query generate a transaction with the evaluation

Table 4.1 – Number of nodes, network density and server type relation for the experiments for the distributed approach for all scenarios

Nodes	Low density	Medium density	High density	EC2 type
20	365x365	316x316	258x258	t2.medium
30	447x447	387x387	316x316	t2.large
40	515x515	447x447	365x365	m5.large
50	577x577	500x500	408x408	m5.xlarge

of the local property. Once the query and parameters are met, the consensus is triggered. The leader elected validates the global property of the query. Finally, it assembles a block and sends it to the network. There is a wait time of 5 seconds, and new random node triggers a query. This is repeatedly done for four consecutive times. Afterward, all the procedures stop and the test case is over.

Scenarios

For all the scenarios, we define three different level of network densities, namely low, medium and high. The low density refers to at least 1.5 nodes every 100m×100m. The medium density refers to at least 2 nodes every 100m×100m. Finally, high density refers to at least 3 nodes every 100m×100m. It is worth mentioning that low density or high density could refer to more extreme cases, but we intend to experiment with different densities. Based on these densities, we define the network size for each number of nodes. The number of nodes determines the type of Amazon EC2 instance where the emulation is running. This is due to the fact of the computation power required by NS3 as the number of nodes and number of packages to simulate increases. The parameter relationship between the number of nodes, network size and Amazon EC2 instance type for the different node densities is summarized in Table 4.1. We define four scenarios to be emulated, which are summarized in Table 4.2. The first scenario analyzes the efficiency of our approach by running the test case with a different number of nodes, different network sizes according to different densities, and different timeout for the raft process with a fixed speed and fixed mobility pattern. The second and third scenario analyze the efficiency as well with the previous parameters for a number of nodes and network size for different densities but having two different speeds and two different mobility patterns. Finally, the fourth scenario compares our results with other approaches. Each scenario runs the previously defined test case through 200 cycles to ensure accurate and consistent results. Therefore, each data point of each graph represents approximately 800 runs of a query, meaning that each graph represents 9,600 runs of a query. It is fair to assume that each graph contains sufficient data points to ensure the quality of the measurements.

Table 4.2 – Scenarios summary for the experiments for the distributed approach

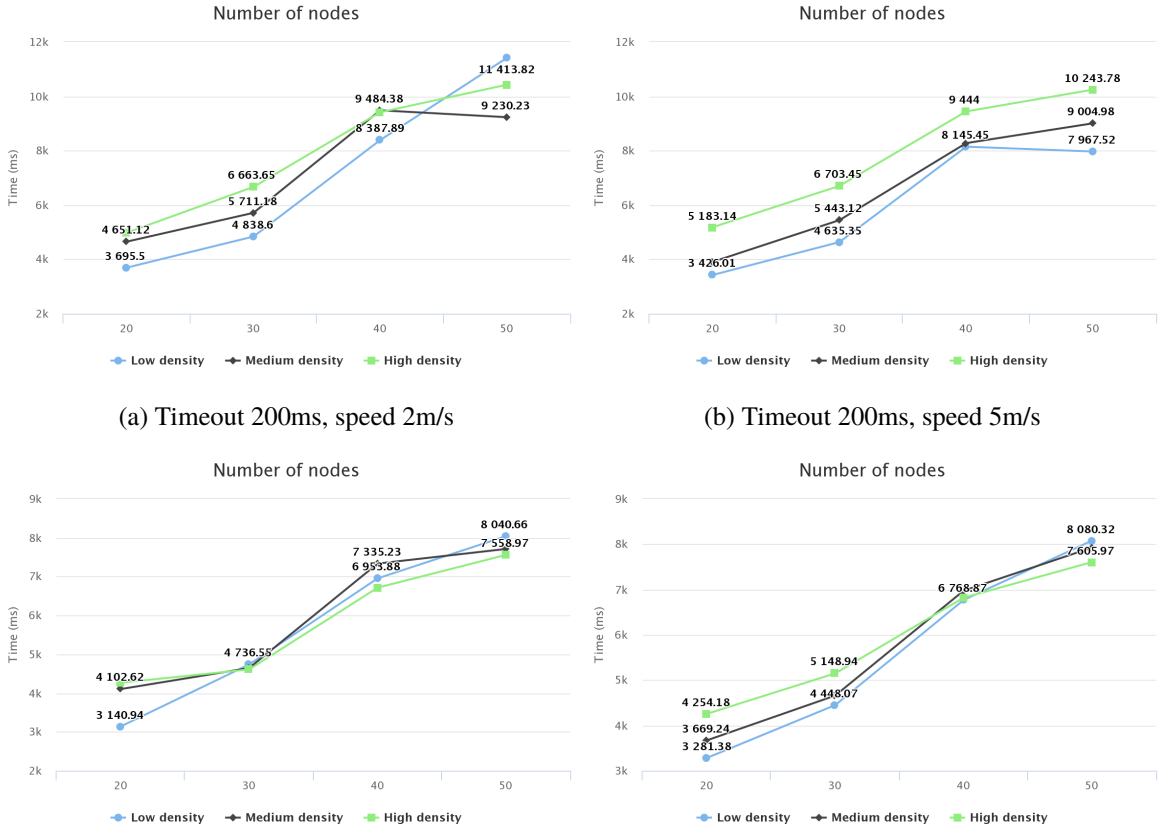
	Scenario 1	**Scenario 2**	**Scenario 3**	**Scenario 4**
Timeout	200ms and 300ms			200ms
Nodes	20, 30, 40 and 50			
Density	Low, medium and high			Medium
Speed	2m/s	2m/s and 5m/s		2m/s
Mobility	RWP		RWP and RandomWalk	RWP
Approach	Blockchain			Treesip, DHYMON and Blockchain

4.3.3 Experimental Results

Results varying the number of nodes, node density and timeout

For this scenario, the intention is to emulate all the combinations defined in Table 4.1. For each pair of number of nodes and network size, we evaluate two timeouts (200ms and 300ms) for the raft process but use a fixed speed of 2m/s and a fixed mobility pattern of random waypoint model. This scenario is designed to test the time and monitoring efficiency of the overall approach. Also taking into account the resources that are being used, we analyze the number of packets generated by our approach. The results for the first timeout of 200ms are summarized in the following way: the results for the time to complete the query are in Figure 4.3a, the results for the accuracy of our approach are in Figure 4.4a, and the results for the number of messages generated by the raft leader election process are in Figure 4.5a and the results for the number of messages generated by the router module are in Figure 4.6a. The results for the second timeout of 300ms are summarized in the following way: the results for the time to complete the query are in Figure 4.3c, the results for the accuracy of our approach are in Figure 4.4c, and the results for the number of messages generated by the raft leader election process are in Figure 4.5c and the results for the number of messages generated by the router module are in Figure 4.6c. Looking at the results, the first apparent remark we can make is that the query completion time increases as the number of nodes increases. Then if we look at the message counts for raft, we notice that a higher number of messages the more nodes there are in the results. This is an interesting but obvious result since it means that the more nodes there are in the network, the most work (packets) and time it takes for the consensus protocol to elect a leader.

On the other side, the router module message count, reflects a steady increase for both cases, since the more nodes there are, the more broadcasts there are and the small fluctuations are due to resend mechanisms to ensure the packet delivery or due to fragmentation problems. Then we can observe some more interesting results for the query completion time, which is faster when the timeout is higher. This is not so obvious, but this is because a timeout of 200ms is too small for the needed interactions to complete an election process compared to the timeout of 300ms. Therefore, there are more problems and re-elections during the election. In a more profound analysis for the completion time, including



(a) Timeout 200ms, speed 2m/s

(b) Timeout 200ms, speed 5m/s

(c) Timeout 300ms, speed 2m/s

(d) Timeout 300ms, speed 5m/s

Figure 4.3 – Query complete time results for random waypoint mobility model for the distributed approach

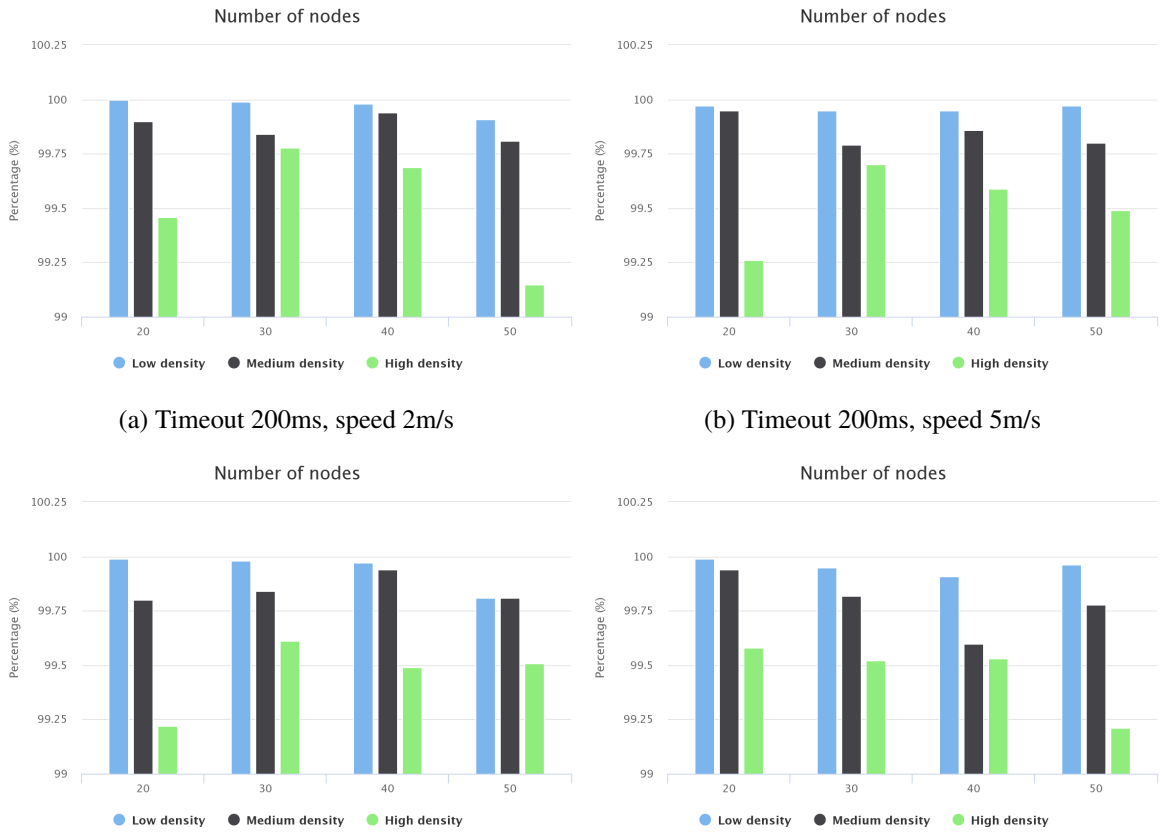
Table 4.3 – Statistical analysis for the query completion time at a speed of 2m/s for the scenario 1 and 2 for the distributed approach

Nodes	Minimum	Maximum	Average	σ
20	1,495	58,743	4,129.33	4,936.19
30	497	58,998	5,341.81	7,250.71
40	737	59,753	8,759.20	10,558.74
50	430	59,997	9,415.15	10,278.96

results for both timeouts, we can refer to Table 4.3. Another highly promising result is the accuracy of the approach. As we can see in Figure 4.4a and Figure 4.4c, the accuracy of the approach ranges from 99% to 100%. This result is significant since it means that although several MANET inherent problems may occur, such as fragmentation, our approach converges to an accurate result. It is worth mentioning that each figure contains a total of 7,500 emulations. The standard deviation of the accuracy results for the timeout of 200ms is 1.90 and for the timeout of 300ms is 2.31. These results make the approach highly effective despite the number of nodes, the density, size of the network, and the selected timeout.

Results varying the number of nodes, node density and timeout for different node speed

For this scenario, the intention is to emulate all the combinations defined in Table 4.1. For each pair of number of nodes and network size, we evaluate two timeouts (200ms and 300ms) for the raft process



(a) Timeout 200ms, speed 2m/s

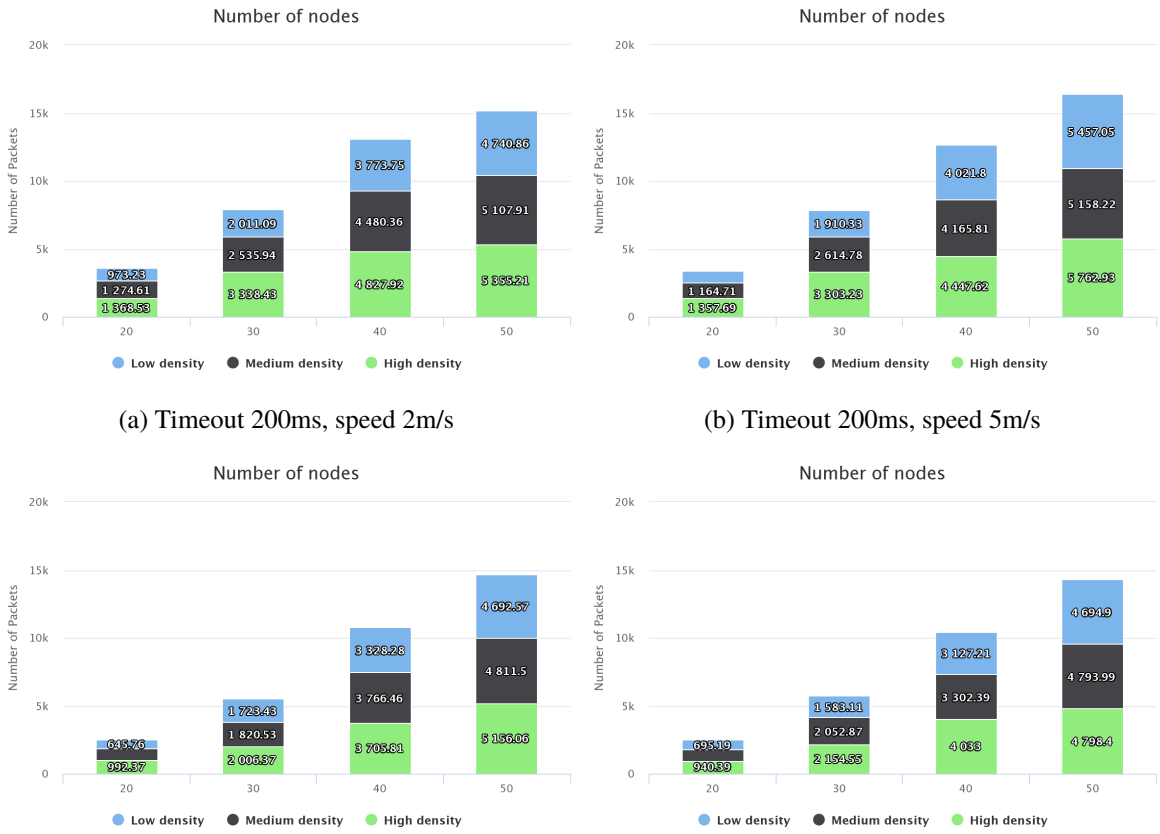
(b) Timeout 200ms, speed 5m/s

(c) Timeout 300ms, speed 2m/s

(d) Timeout 300ms, speed 5m/s

Figure 4.4 – Accuracy results for random waypoint mobility model for the distributed approach

but use two speeds of 2m/s and 5m/s and a fixed mobility pattern of random waypoint model. This scenario is designed to test the time and monitoring efficiency when the speed is different. Also taking into account the resources that are being used by analyzing the number of packets generated by our approach. The results for both timeouts at both speeds are summarized in the following way: the results for the time to complete the query are in Figure 4.3, the results for the accuracy of our approach are in Figure 4.4, and the results for the number of messages generated by the raft leader election process are in Figure 4.5 and the results for the number of messages generated by the router module are in Figure 4.6. From these results, we can confirm the similar behavior in comparison to our previous results. This means that for the four different metrics measured, we can observe similar patterns. For instance, the time for the query to complete is proportional to the number of nodes. There is no clear better or worst time when comparing the two different speeds. In a more in-depth analysis of the completion time, including results for both timeouts for the speed of 5m/s, we can refer to Table 4.4. Comparing Table 4.3 and Table 4.4, we can observe in more detail the similarities. In both cases, we have extreme cases as the minimum and maximum suggested. Based on the standard deviation, we can conclude that for both speeds the average query complete time can vary more as the number of nodes increases. Once again, this makes sense since trying to get to a consensus becomes more difficult the more participants there are. Therefore, it is logical that as the number of nodes increases, the variability of the completion time increases as well. Another impressive result is the accuracy of the result. We can observe that independently from the speed of the nodes, the approach can converge with high accuracy. The standard deviation for all of the accuracy results of a speed of 5m/s, for the timeout of



(a) Timeout 200ms, speed 2m/s

(b) Timeout 200ms, speed 5m/s

(c) Timeout 300ms, speed 2m/s

(d) Timeout 300ms, speed 5m/s

Figure 4.5 – Raft message count results for random waypoint mobility model for the distributed approach

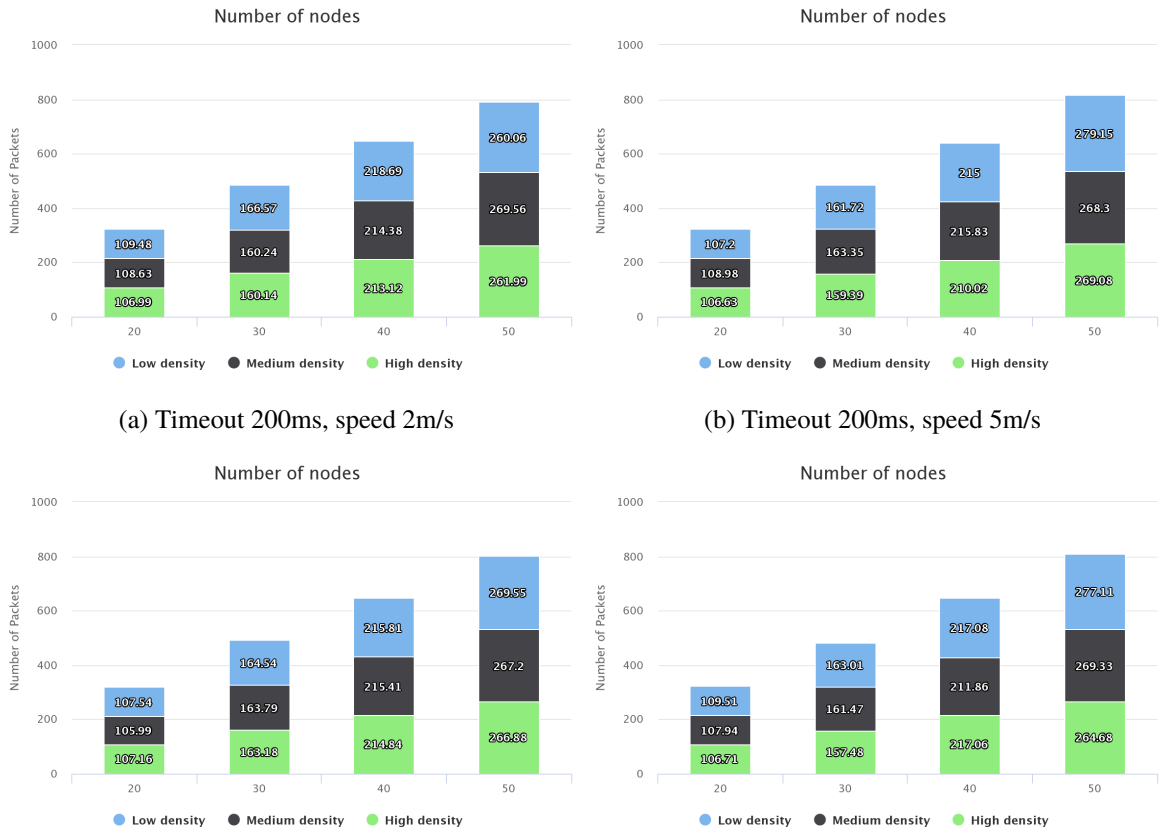
Table 4.4 – Statistical analysis for the query completion time at a speed of 5m/s for the scenario 1 and 2 for the distributed approach

Nodes	Minimum	Maximum	Average	σ
20	1,039	56,136	3,931.39	4,683.64
30	1,547	59,505	5,291.06	7,011.55
40	631	59,784	8,398.48	10,124.50
50	796	59,520	9,003.42	10,089.37

200ms is 1.78 and for the timeout of 300ms is 2.43. Then is fair to assume that the variability of the accuracy is extremely low, therefore making it very accurate and promising for future experiments.

Results varying the number of nodes, node density and timeout for different mobility model

For this scenario, the intention is to emulate all the combinations defined in Table 4.1. For each pair of number of nodes and network size, we evaluate two timeouts (200ms and 300ms) for the raft process but use two speeds of 2m/s and 5m/s and two mobility patterns of random waypoint model and random walk model. This scenario is designed to test the time and monitoring efficiency of the overall approach. Also taking into account the resources that are being used by analyzing the number of packets generated by our approach. The results for the random waypoint mobility model are summarized in the following way: the results for the time to complete the query are in Figure 4.3, and the results for the accuracy



(a) Timeout 200ms, speed 2m/s

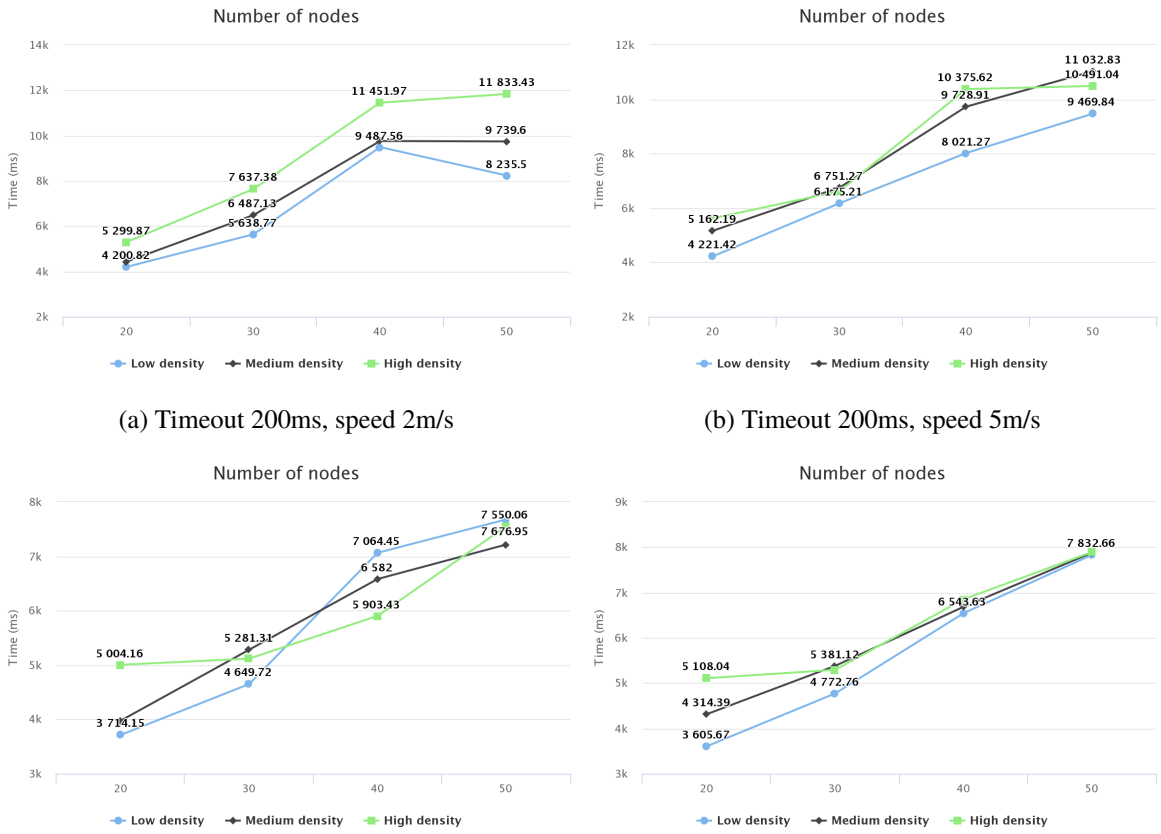
(b) Timeout 200ms, speed 5m/s

(c) Timeout 300ms, speed 2m/s

(d) Timeout 300ms, speed 5m/s

Figure 4.6 – Router message count results for random waypoint mobility model for the distributed approach

of our approach are in Figure 4.4,the results for the number of messages generated by the raft leader election process are in Figure 4.5 and the results for the number of messages generated by the router module are in Figure 4.6. The results for the random walk mobility model are summarized in the following way: the results for the time to complete the query are in Figure 4.7, the results for the accuracy of our approach are in Figure 4.8. The results for the number of messages generated by the raft leader election process and by the router module are omitted since are extremely similar to the results presented in the previous sections. Looking at the results from the query complete time for both mobility models, we can notice that the random walk mobility model results are slightly slower. If we look at it closer, we can analyze the average per number of nodes for all densities and speed for both mobility models. This can be seen in Table 4.5, where it is more evident that indeed is slightly slower. Changing to the accuracy results, we can observe, once again, that the approach is highly effective despite the mobility pattern. Nonetheless, it is important to point out that the accuracy results for high density are more volatile, based on the observation of the standard deviation than the other densities. This is interesting since it shows a scenario where our approach decreases its accuracy but not by much which shows the efficiency of the approach. In a closer look to the accuracy results, the standard deviation for all of the results of a speed of 2m/s, for the timeout of 200ms is 2.57 and for the timeout of 300ms is 2.33. Also, the standard deviation for the results of a speed of 5m/s, for the timeout of 200ms is 3.21 and for the timeout of 300ms is 3.90. These values, compared to the results from the previous sections, show a higher variability due to the mobility pattern. In fact, the lower



(a) Timeout 200ms, speed 2m/s

(b) Timeout 200ms, speed 5m/s

(c) Timeout 300ms, speed 2m/s

(d) Timeout 300ms, speed 5m/s

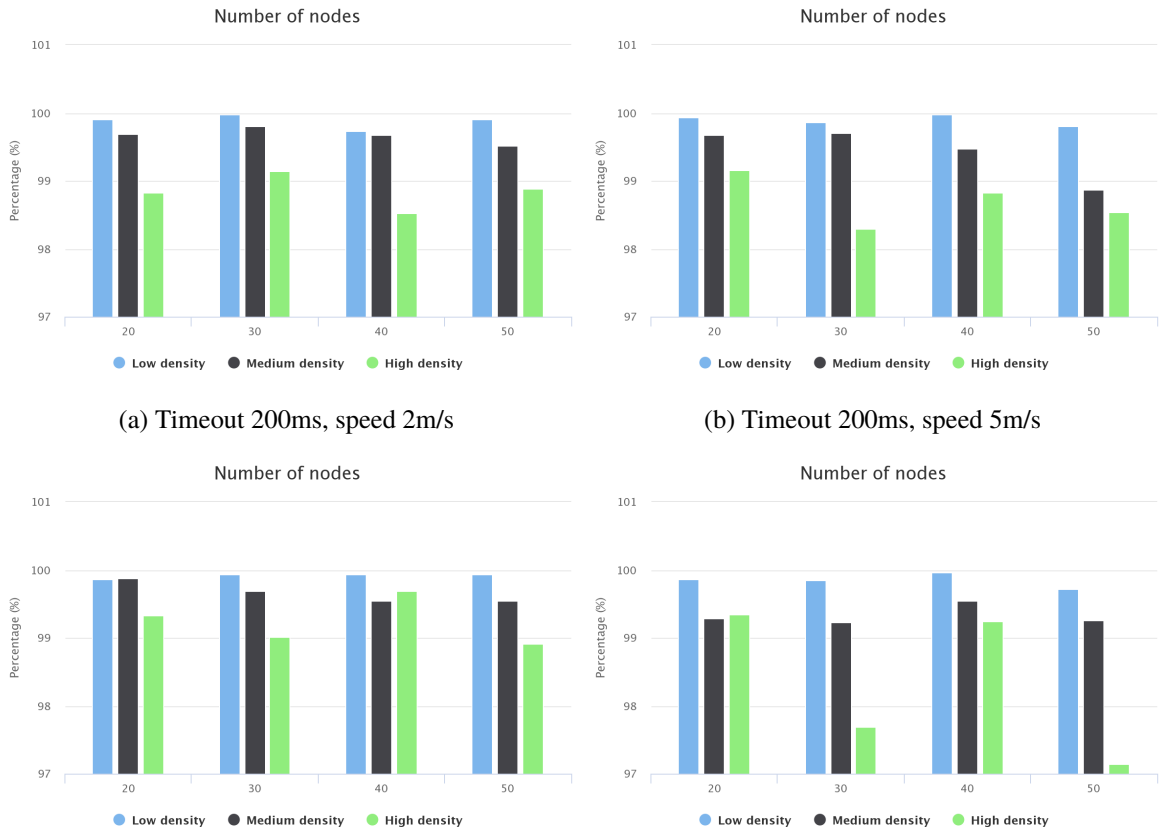
Figure 4.7 – Query complete time results for random walk mobility model for the distributed approach

Table 4.5 – Query complete time for both mobility models for the scenario 3 for the distributed approach

	20 nodes	**30 nodes**	**40 nodes**	**50 nodes**
Random Waypoint	4,301.26	5,800.91	9,998.94	10,489.95
Random Walk	4,847.68	6,738.90	10,256.85	10,768.09

accuracy value is 97.14% (which can be seen in Figure 4.8d), and if you consider that is the scenario of 50 nodes, high density and speed of 5m/s, it is still quite accurate and efficient from our perspective. On the other hand, the raft message count results show an increase as well, it is a small increase, but it tells that due to the mobility, the consensus algorithm is having more difficulties trying to converge on a leader. This is expected since this is correlated to the query complete time, and for the random walk mobility model these values were higher. Setting aside the small increases in the results, we can confirm that our approach works efficiently, independently of the timeout, speed, network size, number of nodes and mobility model.

Aside from the results we have analyzed, it is worth mentioning some other important remarks from the results. We also analyzed the query propagation time and the number of hops for the query. It would be expected that the more nodes, the more it takes to propagate and even more hops. All of our emulations confirm this and also stated by the work of Drabkin et al. (Drabkin et al., 2007). Therefore, it is expected that the query takes from 2 to 5 hops to reach the majority of the network.



(a) Timeout 200ms, speed 2m/s

(b) Timeout 200ms, speed 5m/s

(c) Timeout 300ms, speed 2m/s

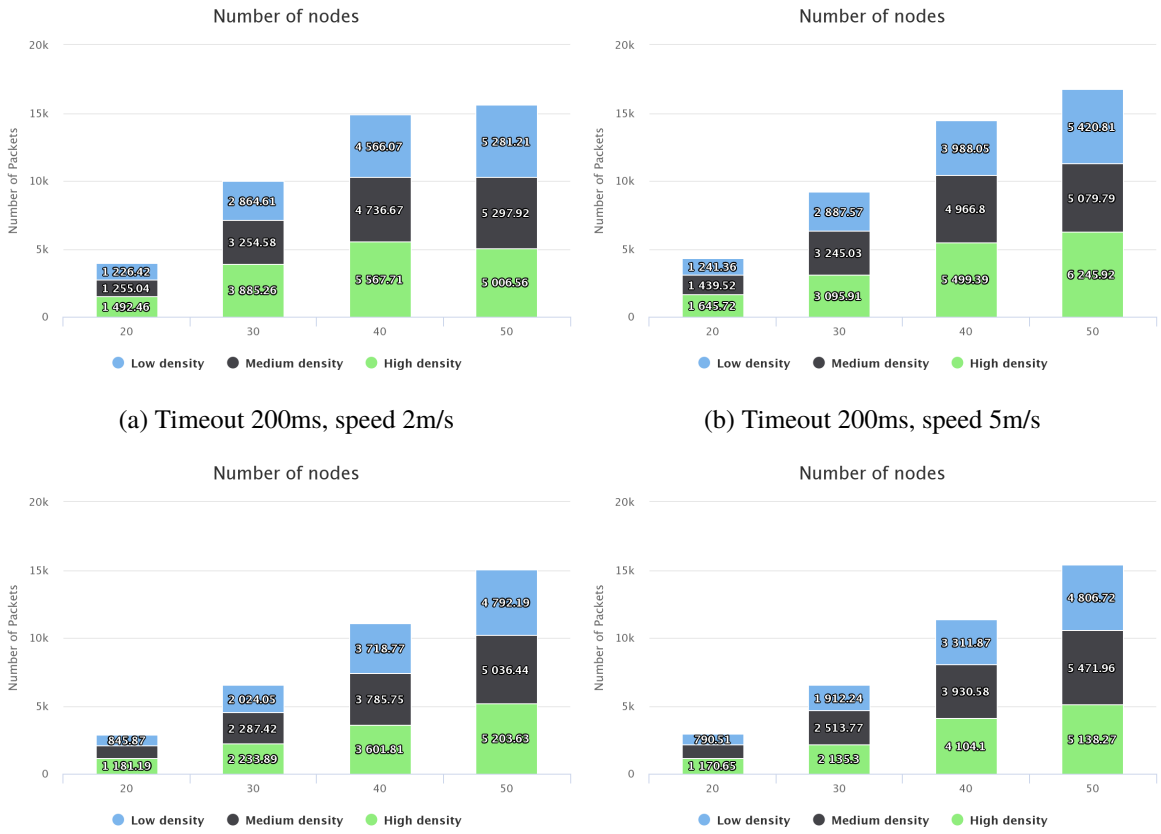
(d) Timeout 300ms, speed 5m/s

Figure 4.8 – Accuracy results for random walk mobility model for the distributed approach

First, we can find the query propagation results in Table 4.6. In this, we can observe that contrary to the query completion time results, the random walk is faster in most of the cases to propagate the query and is less volatile according to the standard deviation. This is interesting since it means that the mobility model affects the consensus algorithm but the query propagation or message exchange is faster. The other obvious remark is that the more nodes there are, the more it takes to propagate the query or any message. On the other hand, we can find the query hops results in Table 4.7. These results seem simple, but it is quite remarkable since it reflects the results from the previously mentioned work of Drabkin et al. (Drabkin et al., 2007). Where they affirm that any broadcast covers a network in 2 - 5 hops approximately, which is reflected in this analysis. Most important of all, the variability of the data is quite small.

Table 4.6 – Statistical analysis for the query propagation for the scenario 3 for the distributed approach

	Random Waypoint		**Random Walk**	
Nodes	**Average (ms)**	σ	**Average(ms)**	σ
20	24.10	7.17	25.75	7.25
30	53.21	23.66	49.81	12.16
40	87.03	55.86	82.98	48.38
50	100.36	89.21	86.20	71.95



(a) Timeout 200ms, speed 2m/s

(b) Timeout 200ms, speed 5m/s

(c) Timeout 300ms, speed 2m/s

(d) Timeout 300ms, speed 5m/s

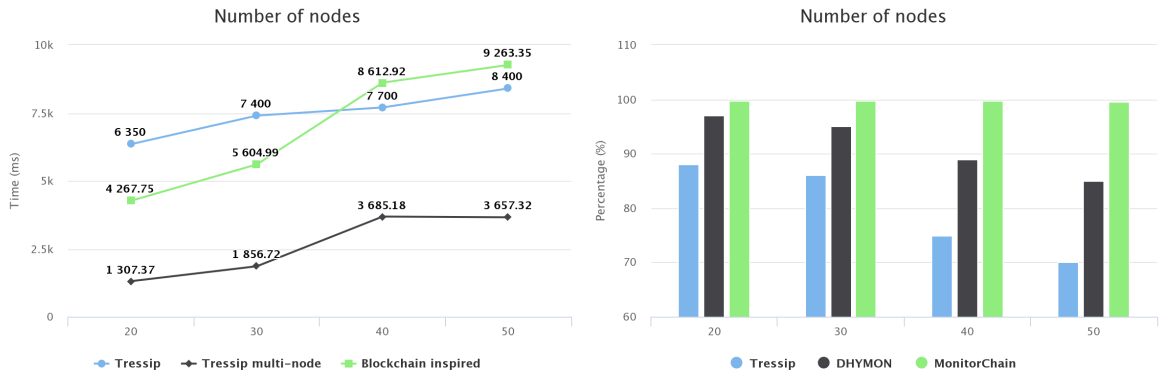
Figure 4.9 – Raft message count results for random walk mobility model for the distributed approach

Results from comparing the Blockchain-inspired approach against Treesip and DHYMON

For this scenario, the intention is to compare the previous results with the approaches of Treesip (Alvarez et al., 2017b) and DHYMON (Alvarez et al., 2017a). Treesip proposes a decentralized monitoring architecture, which combines gossip and hierarchical mechanisms to achieve its final result. DHYMON proposes a semi-distributed monitoring architecture, where the final result is distributed between two nodes. They also rely on gossip and hierarchical mechanisms to compute a result. It is worth mentioning that although both results have shown promising results, neither of them is capable of monitoring a complex function like a global property. Both of them can provide a global view of local properties by performing an aggregation or average among the results, but they are not capable of computing a result that relies on more complex definitions. We can find the results of the convergence

Table 4.7 – Statistical analysis for the query hops for the scenario 3 for the distributed approach

	Random Waypoint		**Random Walk**	
Nodes	**Average**	σ	**Average**	σ
20	2.96	0.87	3.43	1.00
30	3.52	0.96	4.21	1.17
40	4.03	1.00	4.82	1.41
50	4.82	1.18	5.88	1.66



(a) Convergence Time

(b) Accuracy

Figure 4.10 – Convergence time and accuracy for the comparison between other proposals for the distributed approach

time and accuracy in Figure 4.10. The convergence time is defined as the time it takes to the approach from the start of the query to the end of the result. In our case, it is the query completion time, but we use the term convergence time for comparison purposes. The accuracy is defined as the validity of the final result. In our approach, this is calculated by each node and then broadcasted to the network. This means that the network by itself can determine the accuracy per query and therefore per block at runtime. In the approaches we are comparing, an outside analysis of the logs did this. Both approaches logged how many results were they able to aggregate, and the later analysis would consider this value and the number of nodes to calculate the accuracy ratio. The results regarding the convergence time are shown in Figure 4.10a. For the other approaches, although there is not a statistical analysis of the results, they do affirm that they emulated their test cases multiple times to ensure the veracity of the results. The DHYMON approach seems to be the most efficient in term of time, but it has not been tested with different densities and mobility models. The Treesip approach seems to be the overall slowest, and our approach is somehow in the middle. The results regarding the accuracy are shown in Figure 4.10b. This is where it gets interesting, although our approach is not the fastest, it is by far the most accurate. We can see that the Treesip has the worse accuracy ranging from 88% to 70% and the DHYMON ranges from 97% to 85%. The DHYMON has a more steady accuracy but is not as consistent as the 99% of our approach. It is also worth mentioning that the other approaches have not conducted tests to varying the densities of the networks, which could diverge the results. This shows promising results and promising future works for our work given that the experiments are done in a wide range of varying emulation properties. It seems that the other approaches have tried to increase the accuracy, but it always decreases while the number of nodes increases. In our case, the accuracy is outstanding but the time could be improved. So far we know that is dependent on the consensus algorithm.

4.4 Summary

In this chapter, we present our distributed approach for monitoring MANETs which comes from a blockchain-inspired idea. The proposal from our previous approach is designed to monitor simple

functions, but the approach from this chapter focuses on distributed and complex functions. For this, we defined the mathematical support to model a message, candidate event, local property and global property. From this definitions, we can model simple functions like the average CPU. Moreover, we are also able to model complex functions like the integrity of a communicated packet or even the medium transmission time. Both of this properties strictly rely on a global view which of course relies on local views. Then we defined our supporting network procedures to achieve our architecture. For this, we defined a query, transaction, block and consensus procedure. The query procedure disseminates a query which is structured based on our previous definitions. The transaction procedure is the reception of the query in each node. The query is analyzed by the node and determines if it contains enough information to reply a local result. This reply is structured in a transaction, and it is broadcasted to the network. When the transactions are enough to answer a global view of the query, the network triggers a consensus procedure. The consensus is the procedure that helps the network to agree on the global view. Through this procedure a leader is elected by the network, this node computes the global view of the query which is assembled in a block. Then the block is broadcasted to the network and finally verified by the receiving nodes to complete the query. The results obtained were exciting and promising for future works. We achieved an accuracy above $\approx$95% for a different number of nodes, node densities, node speeds, mobility models and timeouts. The convergence time was good, although not as efficient as the accuracy. This can be improved by enhancing the consensus algorithm. But regardless of this, the overall results showed outstanding performance of our approach.

For this chapter, we approximately ran 76,800 emulations. This allowed us to achieve a significant amount of data generated by our emulator and high confidence in the credibility of the results. And all of this emulations were done in less than 24 hours. We believe that our emulator truly shines by itself and provided tremendous support for our experiments. For this, we describe the set of tools that allowed us this in the following chapter.

5

Tools and Implementations

Contents

In Chapter 3 we proposed a hybrid-based monitoring approach. During its development, we decided that we needed to provide a reliable and robust set of experiments to prove the efficiency of our approach. This led us to survey the simulation and emulation for MANETs literature. Among

the different studies, we found the work of "Manet Simulation Studies: The Incredibles" (Kurkowski, Camp, & Colagrosso, 2005), which served as a firm bedrock of what we wanted to achieve. This study states that all experiments must be repeatable, unbiased, rigorous and statistically sound for them to be credible. The repeatability of a research study is crucial, and it can be said that is one of the main pillars of science in general. In our context, it refers to provide enough information for other researchers to re-run the same experiments. This includes providing the simulator and version, the operating system used, configurations and any other code used for the experiments. The experiments need to be unbiased. This is achieved by relying on different scenarios using random values and any initialization bias. The rigorous of the study needs to be attained by varying as many parameters as possible to stress as much as possible the protocol under review. Finally, the scientific procedure needs to statically sound. This is done by executing the experiments multiple times to provide a degree of confidence level in the results. This also means to take into account any initialization bias and statistical assumptions made for the study.

Using the work of Pawlikowski et al. (Pawlikowski, Jeong, & Lee, 2002) as a starting point, the network literature surveyed from the simulation perspective is questionable. In the sense that, the studies do not often provide the sufficient information about how the experiments were conducted. This work was done in 1999 and focused on networks in general. Therefore, one might argue that due to the time and the technology it might not have been as easy as it has become in the recent years to share this kind of information. But the work of Kurkowski et al. (Kurkowski et al., 2005) concludes that their findings are discouraging for the time of the study. They conclude that similar to the conclusions from Pawlikowski, the pattern is consistent even years later. Sadly this pattern has been consistent over the recent years. Other studies have confirmed this as the one from Sarkar et al. (Sarkar & Gutiérrez, 2014) and the work of Cavalcanti et al. (Cavalcanti, de Souza, Spohn, Gomes, & Costa, 2018). Both of these studies conclude that in the recent years the number of studies has grown, but the credibility is still questionable. All of these conclusions led us to focus on doing the experiments as it is expected from the community with the corresponding credibility.

Based on this, we provide a set of tools and implementations to ensure the credibility of our experiments.

- We implemented an emulator based on the network simulator (NS3) and the container technology of Docker. This provided us the versatility of NS3 for modeling mobility patterns, realistic transmissions with signal decays and also the ability to share our configurations via C++ code. Then Docker provided the tools to encapsulate the implementations in a real OS-virtualized environment connected to NS3.
- Relying on our emulator, we built an Amazon Web Services orchestrator. This allowed us, to run multiple emulators simultaneously in an automated, repeatedly and efficient manner. Then, we could afford to run more variants of the same network multiple times. Although this brought some complications. Eventually, the amount of data generated was difficult to parse and analyze due to the quantity of it.
- Based on the problematics due to the increasing amount of information generated, we built an emulator web output. This led to an improved generation and analysis of our results. Once again,

this allowed us to run even more network variations and efficiently collect, parse and analyze the results.

- Finally, for each experiment that we ran, we developed the proposals using the language Go. To ensure the credibility of our approaches we have all the different implementations in public repositories.

This chapter is divided as follows: In Section 5.1, we give the basic concepts and notions about the simulation concepts. Then, in Section 5.2.1, we describe our emulator, alongside all the different achievements and difficulties we had. Later, in Section 5.2.2, we outline the orchestrator that allowed us to get a vast amount of results in significantly less time. Afterward, in Section 5.2.3, we narrate the difficulties we had with too many data being generated, which led us to build a tool to improve the analysis. Finally, in Section 5.3 and Section 5.4, we detail the different implementations that were developed to finally measure the efficiency of our proposal with the help of the different stages of our emulator.

5.1 Preliminaries

The need to conduct experiments in the networking literature has been a fundamental interest. Therefore, the need for improvements, enhancements and novel tools is part of this community. One of the most critical aspects of network simulation is the versatility of running experiments without the need for the physical devices. This allows researchers, professionals, and educators to test, evaluate and measure any network protocol or implementation.

5.1.1 Network Simulators

Multiple network simulators are used in the literature. Based on the survey of Kurkowski et al. (Kurkowski et al., 2005), the most prominent simulators are NS-2[1], self-developed, GloMoSim, QualNet, OPNET, CSIM, and MATLAB. Based on the work of To et al. (To, Cano, & Biba, 2015), the most prominent open source tools for simulation or emulation are NS-2, NS-3[2], and OMNET++[3], and the most prominent commercial tools are QualNet[4] and OPNET[5]. From these works, we can assure that NS-2 and NS-3 seem to be the most prominent network simulators. NS-2 is known to be time-consuming, not modular and not efficient due to its high computing resource utilization. On the other side, NS-3 is more flexible, modular and efficient. It is worth mentioning that it is far from perfect or performant, but it is good enough for most use cases. This is why it has become a trendy choice in the network literature. NS-3 is a discrete-event network simulator built on C++. It can simulate wired and wireless networks. For wireless scenarios, multiple NS-3 modules can help with random

[1]https://www.isi.edu/nsnam/ns/
[2]https://www.nsnam.org
[3]https://www.omnetpp.org
[4]https://web.scalable-networks.com/qualnet-network-simulator-software
[5]https://www.riverbed.com/gb/products/steelcentral/opnet.html

start point for the nodes, random mobility models and signal decay models. It provides the necessary modularity to build your custom libraries and adapt them to the simulator. Although this is useful, it is still highly dependent on C++ or Python through the use of a specific module. In both cases, it is needed to know in depth the behavior of the emulator which makes the learning curve somehow steep. It is worth mentioning that, this is a very good solution to simulate a network, but it is still done in a controlled environment. Another aspect in favor of NS-3 is their community. It provides proper documentation, active development and a lot of useful resources for its utilization. Therefore, it can be said that NS-3 is one of the actual favorite network simulators due to its many features.

5.1.2 Docker: The Container Movement

For a long time, using monolith applications and servers was the only possible solution. Aiming to maximize the resource utilization, there were successful attempts to do this like virtualization. Virtualization can be briefly described as the allowance of multiple virtual machines to be co-located on a single physical machine (Bari et al., 2013). Virtualization goes a long way back from 1960. There are different types of virtualization, the first type was done by hardware and later by software. The hardware virtualization can be subcategorized as full virtualization and paravirtualization. Full virtualization is a complete simulation of the actual hardware, and the operating system runs unmodified. Meanwhile, for paravirtualization, the hardware is not fully simulated, and the operating system needs the corresponding modifications to run with this. A more recent type of virtualization is operating-system-level, also known as containerization. Although this one, it was first proposed in 1982, it got popularized until recent years with Docker(released in 2013).

This type of virtualization is done at the kernel level. It creates and manages user-space instances in an isolated way. This isolated user-spaces are called containers. Therefore, this is an improved virtualization since instead of installing multiple operating systems, you can run isolated spaces on top of the same operating system. This allows a higher efficiency in resource utilization and due to this its recent popular broad utilization. It is worth mentioning that due to the use of containers, there has been a significant paradigm shift from monolith applications to microservices and serverless applications. This trend tries to make more modular and isolated applications to maximize the use of this type of virtualization. The more prominent implementations are chroot (1982), sysjail (2006-2009), LXC (2008) and Docker[6] (2013) among others. From the different implementations, Docker became the most popular of all since it is focused on the ease of usability. Aside, they have made significant efforts to provide an efficient open source solution and as well as a robust enterprise edition. All of this, alongside proper documentation and a proactive community has led to an excellent containerization solution.

[6]https://www.docker.com

5.2 Designing a Scalable Network Emulator

5.2.1 NS-3 Docker Emulator

To test our proposals, we first evaluated the option of using NS-3 and C++ to develop our implementation. Although we concluded after multiple weeks that NS-3 although is good to simulate networks, it is not the best option to build a custom complex applications. Therefore, we needed something a little bit more abstract that would give us the flexibility to develop in a different programming language. We started considering other options around NS-3, this search led us to a proposal of using NS-3 and Linux containers (LXC) (NS-3, 2010). The idea was that NS-3 simulations could be connected and integrated with real hardware to make the simulations more realistic. This is an exciting idea that resonated in the network simulation literature. The work of Handigol et al. (Handigol, Heller, Jeyakumar, Lantz, & McKeown, 2012) states that a research paper should be easily reproducible by clicking on a button to replicate all the results. This is a utopia scenario, and although we are far from achieving it, there are significant efforts toward that direction. Therefore, this paper analyzes the feasibility, outcomes, and experience of using containers in the context of network simulation. Their conclusion is that is highly feasible and that, in their case, the experiments were reproducible by a different person or team in less than a day. Then the work of Calarco et al. (Calarco & Casoni, 2013) analyzes the effectiveness of using Linux containers and network virtualization from the performance perspective. They conclude that they provide an excellent resource utilization. They also found that even though it can be useful when analyzing fast speed networks, it could have bottlenecks due to the core functioning of the containers.. Nonetheless, the capacity of packets per second for the simulations was extraordinarily high, and it showed promising results for future works. Later the work of Chan et al. (Chan et al., 2014) goes into the domain of using NS-3 and containers for wireless networks. They analyze the specific case for software-defined wireless networks. They conclude that there is a need for improvements in the wireless networks simulations since NS-3 falls short for complex applications and Mininet [7] does not support wireless modeling. Therefore, they provide OpenNet as an alternative trying to connect Mininet (which relies on LXC) and NS-3. Finally, the work of To et al. (To et al., 2015) analyzes the possibility of connecting NS-3 and Docker for MANET emulations. They propose the combination of these two technologies through a network bridge mechanism provided by the kernel. This means that a container is only connected to a software network bridge in the kernel. On the other side, NS-3 exposes each simulation node to a tap interface in the kernel. Finally, an intermediary bridge is created to connect the container bridges to the NS-3 tap interfaces. This last work was the necessary solution we were expecting. Unfortunately, we were not able to find any repository, code or any insights on the internals of this proposal. Given the fact that this proposal seemed to fit what we needed, we decided to implement this idea on our own. We were also convinced that we need to implement it and to follow along with the recommendations for network simulations. This meant we needed to share it publicly and ease the utilization for other researchers to use it.

The NS-3 Docker Emulator can be found in Github [8] and the proper documentation can be found

[7] http://mininet.org
[8] https://github.com/chepeftw/NS3DockerEmulator

on GitHub pages [9]. The resulting emulator architecture can be found in Figure 5.1. The architecture sits on top of any server. The outer black box represents this. In our case, we first tested it on a virtual machine running on top of VirtualBox. Later, for our real experiments, we moved it to an AWS EC2 instance. Then we have three main components, the kernel, Docker, and NS-3. The kernel refers to the Linux kernel where Docker, and NS-3 are running. NS-3 takes care of simulating a wireless mobile network. Each node inside the simulation is connected to a tap interface in the kernel. This means that each node inside the simulation acts as a moving antenna. Docker is in charge of running the containers. Each container is first started as an isolated container, i.e., it has no network connectivity. Inside each container, we could have a standalone or a group of applications. The emulator orchestrates and integrates these components to provide an efficient and accurate MANET emulator. The emulator runs in five phases which are summarized in Figure 5.2. It starts by collecting the base parameters for the emulation, i.e., number of nodes, time for the emulation, timeout for the emulation, size of the network, node speed, node pause if the mobility model requires it and number of emulations. The first phase is to build and run the containers. The containers image is built using the multi-stage build Docker feature, which allows a stage to build the implementation under test and in the second stage to run the executable on top of an Alpine kernel. This makes the image highly efficient with a size of 15Mb. To put it in perspective, the Ubuntu base image is 188Mb, and the CentoOS base image is 172Mb. Then it runs the containers based on the number defined before. After running the containers, it modifies the user space in the kernel to create the external bridge. The external bridge is created with a tap interface in promiscuous mode. The second phase is to compile and run the NS-3 simulation. The first step is to compile the simulator with a specific set of parameters to make it more efficient. This means disabling examples, python support, and optional modules. Otherwise, the simulator can consume many resources and slow down our simulations. Then it compiles our simulation, which defines the nodes, the ad-hoc network, configures the physical layer to rely on real-time models, sets the mobility, configures the kernel tap interfaces and finally runs the simulation. The third phase is the link between the docker external bridge and the NS-3 tap interfaces. This is achieved by creating peer interfaces. During this process, we create the eth0 interface for the container, we assign a MAC address, the IP address and the network segment. After this, the emulation is finally running. The fourth phase is the recompilation of data, wrap up and releasing resources. This is a critical phase, since all the network tap interface, external bridges and peer interfaces need to be freed to re-run the emulation. This required multiple techniques to store the interfaces id to release the corresponding resources later. This also led to multiple problems when running consecutive emulations. This was caused by the inability of the kernel to clean up the resource utilization due to the high creation interfaces interval. Finally, the fifth phase is to repeat the process to run multiple emulations.

The emulator allowed us to run repeatable experiments. It was used in every emulation done and defined in the Chapter 3 and Chapter 4. The emulator was built alongside the experiments of Section 3.3.2. Aside from this, it has been used by other researchers as it can be seen in the Github repository page. This interest from other fellow researchers showed us that indeed there is a need for a more updated network simulator. Not only that, but it is required to provide the proper documentation, code, and tools for them to build on top of it.

[9] https://chepeftw.github.io/NS3DockerEmulator/legv1/homelegacy.html

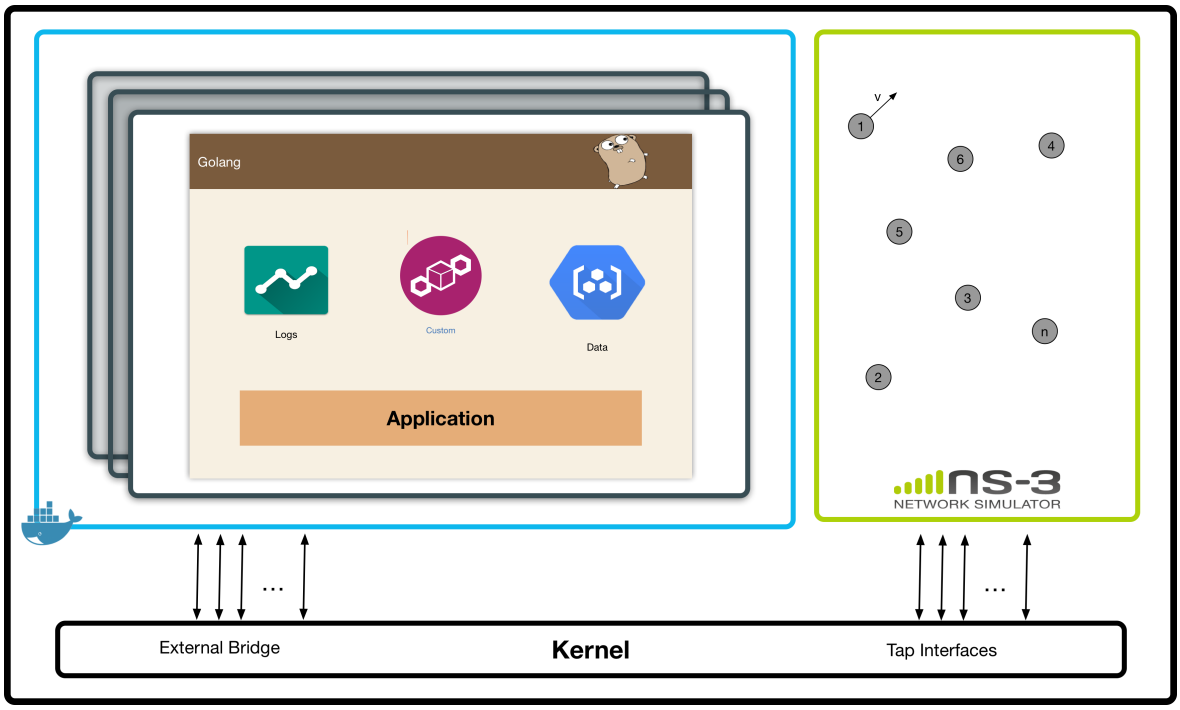


Figure 5.1 – NS-3 Docker Emulator

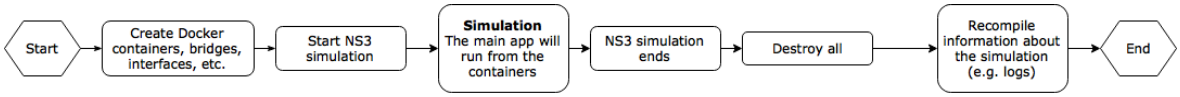


Figure 5.2 – NS-3 Docker Emulator version 1 flow

5.2.2 Amazon Web Services Orchestrator

The NS-3 Docker emulator allowed us to run repeated experiments in a quite organized and efficient way. However, although we were able to run 20 to 50 emulations per scenario, we wanted to run hundreds of time per scenario and as fast as possible. The problem with the emulator was that each experiment was run in a sequential way. To increase the number of emulations, we needed to run them in parallel to improve the scalability. Revisiting the work of Calarco et al. (Calarco & Casoni, 2013), they point out that relying on Amazon Web Services (AWS) allowed them to reproduce their experiments easily. Based on this and our own experience, we decided that we could integrate our emulator with AWS Elastic Cloud Computing (EC2) service. AWS EC2 [10] is a service that provides computing capacity (i.e., servers) in the cloud which are flexible, efficient and fair-priced. This led us to the version 2 of the emulator [11] and the AWS orchestrator [12].

First, we improved the flow of our emulator. After multiple runs, it became evident that it was a bad idea to build the containers and to compile the NS-3 simulation every time. On the other side, when we started developing the emulator, it was not clear how we would make it scalable, so this was something we did not include in our design. Therefore in this version 2, we decided to decouple a lot of the existing processes to make them more modular. This process took much time since it meant

[10] https://aws.amazon.com/ec2/
[11] https://chepeftw.github.io/NS3DockerEmulator/
[12] https://github.com/chepeftw/EC2TestAutomator

rewriting a lot of the existing scripts to support the needed modularity. On top of that, we updated multiple scripts from bash and Python 2.7 to Python 3.0. Initially, this also took much time since there were many compatibility issues to work around. Therefore the emulator runs in five phases which are summarized in Figure 5.3. First, the emulator builds the container images. Second, the emulator compiles NS-3 in optimized mode, compiles the simulation and starts the simulation. The third and fourth phases are the ones that changed. The third phase consisted of starting the emulation with the restart of the containers. This means that if the container is running, it will be restarted, and if the container is not running, it will be started. Alongside, the necessary bridges and tap interfaces would be created. The fourth phase consists of recompiling the information of the emulation. This means analyzing the logs or collecting the necessary information to analyze them afterward. Then, it would loop back to the third phase. The emulator knows the number of iterations to run, and this is how it controls how many times to loop. Finally, the fifth phase consists of freeing up the allocated resources. This different flow from the previous version allowed the emulator to be more efficient and reduce the probability of failing. This was because multiple software-defined resources allocated by the kernel were reused. This led to efficiency and substantial time improvements for the emulations in general.

The AWS Orchestrator relies heavily on AWS and a service to work correctly. The fundamental problem with this approach was how can we run different servers with different parameters in a dynamic and automated approach. The solution was to treat the server as a function. This means that we needed a mechanism to pass some arguments. Followed by an internal mechanism to take these arguments and execute the emulator in the corresponding flow. The arguments mechanism was achieved by relying on a tagging feature of AWS EC2, i.e., each server that is created can have a set of key-value pairs as tags. The tags allow the managers of the servers to identify them according to their use or purpose. Therefore, we could create an instance with a predefined set of tags where it was specified the number of nodes, network size, node speed and so on, for that particular emulation. Then, we created a service that starts during the boot of the server and takes immediate control of the server. It starts by pulling the tags and storing them in memory. It is worth mentioning that from inside the server, it does not know this extra information (e.g., instance tags). Therefore it has to pull it from an internal AWS service. From here, the boot service would start the regular flow of the emulator. After the emulations ran, this service collects the emulator information and would upload it to a specific server where it was processed once all the emulations were done. The orchestrator allowed us to run N parallel emulations and helped us to generate an overwhelming amount of information for us to analyze. The orchestrator is illustrated in Figure 5.4. It is shown the schematics of the emulator, but it is now running in an AWS EC2 instance, which is represented by the outer orange box. It shows multiple outer boxes representing the parallel executions of our emulator.

The emulator allowed us not only to run repeatable experiments but to scale them in a virtually unlimited way. It was used in the second set of experiments in Chapter 3 and all of the experiments in Chapter 4. The AWS orchestrator was built alongside the experiments of Section 3.3.4.

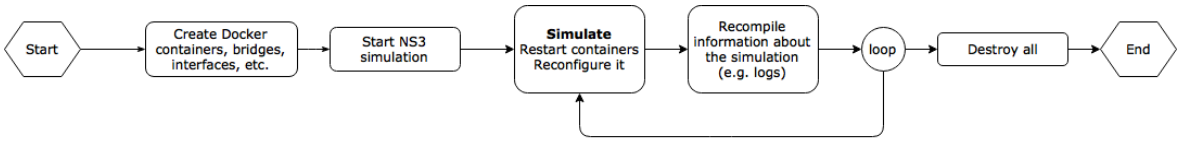


Figure 5.3 – NS-3 Docker Emulator version 2 flow

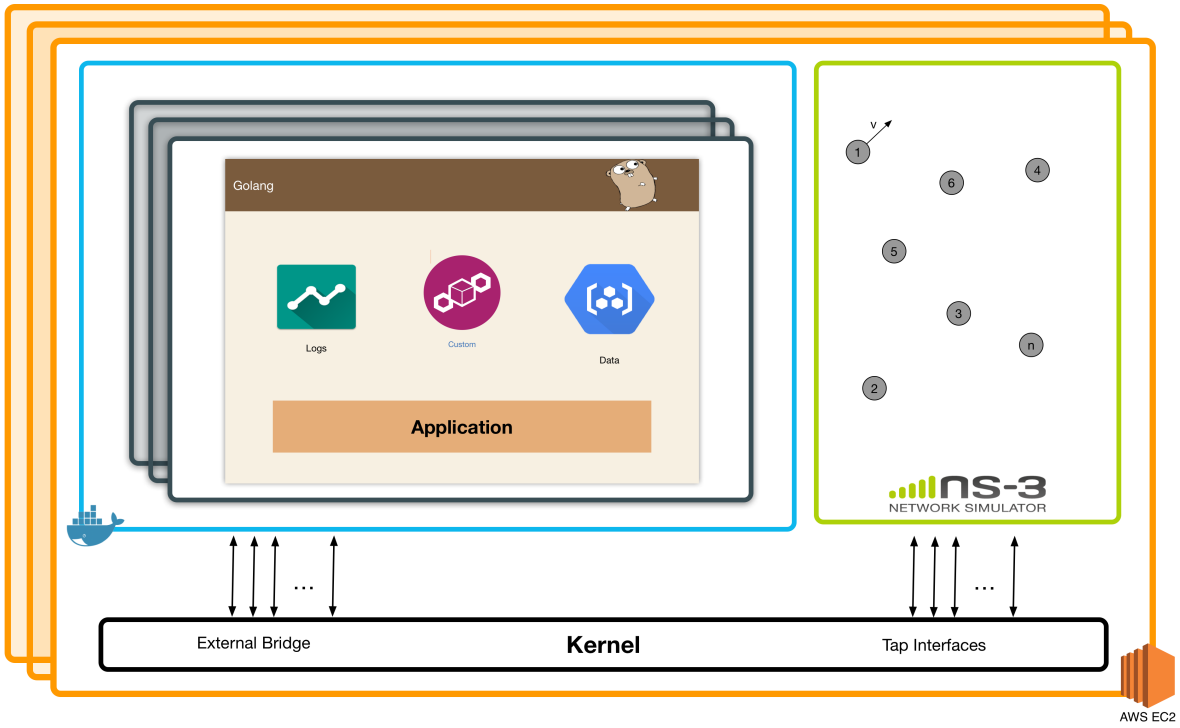


Figure 5.4 – NS-3 Docker Emulator and AWS Orchestrator

5.2.3 Web-based Emulator Data Processing and Output

The AWS orchestrator combined with the NS-3 Docker emulator allowed us to run efficient, repeatable and scalable experiments. Unfortunately, this induced an unconventional problematic, which was it became challenging to analyze and process on real-time the data from the logs of the containers. The other problem was the availability of the data. Therefore, we needed to wait for the emulations to conclude to start processing the data. This lack of immediate visualization of the data limited us to test, measure and improve at a faster pace. Our goal was to run the experiments and analyze the results in real-time to optimize the process to derive conclusions and improvements when necessary. Therefore, to solve this situation, we decided to collect the data of the containers in a database to analyze them faster and efficiently. For this we decided to add a server to collect the data using a database, then a layer to process the data and finally an API to expose the results in real-time. The tool is called Emulator Web Output, which can be found on Github [13].

For this tool, we improved the AWS orchestrator and developed a custom tool. The overall architecture for this can be seen in Figure 5.5. The component of the NS-3 Docker emulator and the AWS orchestrator remains almost the same. The improvements to the orchestrator are represented by the "Logs API" in the figure. The improvement was made in the data collection done by the orchestrator, instead of collecting them locally, it collected them by pushing this information to MongoDB. This process was done after every emulation to allow the analysis in real-time. Therefore, the orchestrator started, then it made the initial setup to run the emulator. Afterward, when it loops through the emulations, it would push the results for each emulation to the MongoDB.

Then the data processing layer was done using MongoDB (v3.2), GoLang (v1.9) implementation, Nginx (v1.15) and Docker (v18.03). In Figure 5.5, it can be seen in the left an AWS instance with the components used for this layer. MongoDB was used as a storage layer running as a Docker container. It was accessed for write operations by the AWS orchestrator and for read operations by the GoLang implementation. The implementation has two main parts, a predefined query structure, and the API. The predefined structure was used to store and fetch the results easily. Based on this, the API queried the database. The query was done to a different collection of the database, depending on the parameters received by the API. The query is done by calculating the average, minimum, maximum and standard deviation value for the queried value grouped by the emulation. This allowed us to get a better analysis on real-time from all the emulations. The Nginx had simple HTML files, one web page that queried the API and then showed the results in a table format. Aside, a second web page that showed the results from the API in a graph.

This tool allowed us to collect, process, analyze and visualize efficiently the data generated by our overall emulator. It might seem simple, but it was of enormous importance to allow us to conduct multiple experiments and analyze them in real-time. The emulator web output was used and built alongside the experiments of Chapter 4.

[13] https://github.com/chepeftw/emulatorweboutput

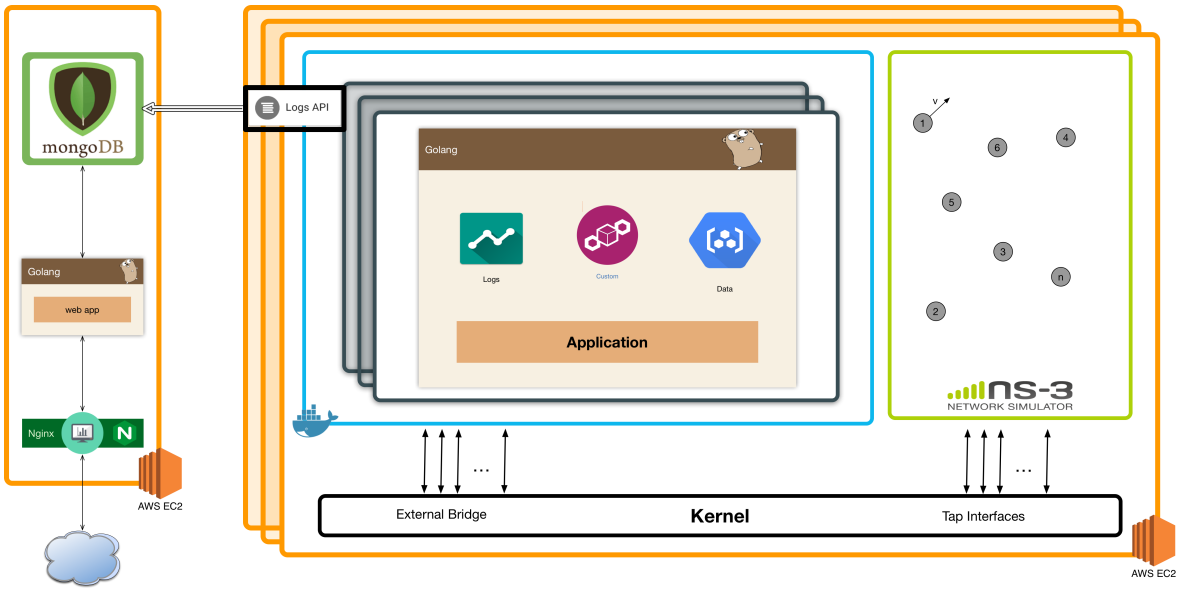


Figure 5.5 – NS3 Docker Emulator, AWS orchestrator and Emulator Web Output

5.3 Decentralized Monitoring Implementations

It is worth mentioning that this implementation had different stages. The first stage was done for the Section 3.3.2, which was without mobility support. Everything was done in a single file since it was simple enough[14]. The second stage was done for the Section 3.3.3, which was with mobility support. This stage was separated into four repositories:

a) Common libraries to share the packet structure and some other common utilities[15]

b) The Treesip implementation

c) The gossip routing fallback protocol[16]

d) A support tool to route packets relying on the routing table with our emulator[17]

Finally, the third stage was done for the Section 3.3.4, which was with multi-root node support. This stage were small improvements over the existing repositories.

5.3.1 Treesip Implementation

The Treesip implementation models the FSM presented in Figure 3.3 and Figure 3.4. This implementation takes care of three main tasks:

[14]https://github.com/chepeftw/Treesip
[15]https://github.com/chepeftw/Treesiplibs
[16]https://github.com/chepeftw/Gossip
[17]https://github.com/chepeftw/DirectMessage

a) Identify the root node decided by an argument received from the emulator and assembling and sending the query

b) The behavior based on the FSM

c) The root node acknowledgment to the emulator when the final result is computed

The emulator calculates each round a different root node. By default, it always started with the first node created and then it changed randomly. This value was written to a config file by the emulator, and the implementation would read it from there. All the nodes would try to see if they were the root node, but only the corresponding root node would respond. This selection was made using the IP address. Once the root node is selected, it assembled the query. The query for the Treesip approach always calculated the average CPU utilization for the time it received the query. This was done intentionally to get a scalar value to work with. After the query is assembled, it is transmitted to the network. Then the implementation would emulate the FSM. Meaning that it would keep a state and it would fire a transition from state to state depending on the previous definition. Based on this definitions, it would send messages or start the corresponding timers to ensure the overall behavior. Finally, the root node outputs and acknowledges the emulator via the logs when the final result is calculated.

5.3.2 Gossip-based Routing Fallback Procedure

This implementation was designed to work as the fallback mechanism for the aggregation procedure of our decentralized approach. This relied on a random basic gossip algorithm. The protocol was reactive to packets. Then it relied first on a hello packet, that is broadcasted. The hello packet has the ID of the packet to be transmitted. This way all nodes can keep track of which messages are being routed. The nearest neighbor answers a hello reply packet. After this, the node that sent the hello packet sends to the replying node the real packet. To avoid loops, we defined that a node would have to wait a small random time to send a second time an already sent packet. For example, a packet with ID 1 is being routed through the network. Eventually, a node replies the hello from another node, it receives the packet and forwards it using the same mechanism. After some hops, due to mobility, the same node could receive a hello message from a different node but for the same packet with ID 1. Therefore, in this case, to avoid loops between the nodes, this node waits a small random time to give the opportunity to another node to route the packet. This is done to try to maximize the node utilization and to avoid that the same nodes always forward the packets. Every time a node received a packet, it checks first if the packet is for him and if not it proceeds with the described behavior.

5.3.3 Routing Support Tool

Our emulator has had a network limitation due to how it is built. The problem is that Docker creates by default an internal software-defined network for the containers. For our emulator to rely on the NS-3 simulation instead of this internal network, we have to start the containers in a mode without a network. Then we manually create in the kernel (not through Docker) the corresponding network definition

for each container. This creates the problem that the container does not recognize the network at the application level, but it does recognize it at a low level. For example, if a program tries to send a packet relying on the operating system, the packet never gets sent because the OS thinks it does not have any networking interface. Meanwhile, if it tries to send a packet programmatically through a socket, this packet gets sent to the network, and other nodes receive it. Another example is the OLSR routing protocol behavior during the experiments detailed in Section 3.3.3. When we ran OLSR inside our nodes, the routing table was updated, and each node was able to see each other through the OLSR protocol. Once again, the OS was not able to act on top of the routing table since according to him the container had no networking capabilities. We analyzed different solutions to this problem. One option was to tell docker the corresponding network interface, but to do this it was required to create a network plugin for Docker. The problem was that the documentation for network plugins at the time (early 2016) was inexistent and only recently (early 2018) has begun to exist and it is improving. Nonetheless, it is still unclear how to accurately create a custom one. Only a few partners have access to more Docker resources and have created their own. The second option was to do some update inside the OS to make it aware of the networking capabilities. This required some Linux commands that are not supported in OS-virtualization since they modify some essential files from the kernel which are not available inside a container. The third option was to create an intermediary tool that reads the routing table, parses it and sends a packet accordingly to this. Basically doing the OS role for routing packets.

Due to time constraints, we chose the third option as this repository was the result of it. The tool runs OLSR in the node. Then the OLSR program starts doing the reactive protocol handling. So it is continuously updating the routing table. Then the tool reads the routing table from the OS. Parse all the available nodes, available routes, and remaining information. Based on this it selects a node to forward the packet. Then it would forward it, and the same tool running on the destination node would capture the message and forward it internally to the Treesip implementation. All of this was done as a workaround of known problems of our emulator that required too much time to solve correspondently. Finally, this tool allowed us to conduct the comparison between our gossip fallback routing mechanism and OLSR routing protocol.

5.4 Distributed Monitoring Implementations

The implementation is separated into four different repositories. Each repository represents a module of the complete implementation, it is separated into ledger[18], miner[19], router[20] and raft[21].

5.4.1 Ledger module

The ledger module takes care of three main tasks:

[18]https://github.com/chepeftw/LedgerMANET
[19]https://github.com/chepeftw/MinerMANET
[20]https://github.com/chepeftw/RouterLedgerMANET
[21]https://github.com/chepeftw/raft

a) Create and send the query

b) Generate a transaction

c) Validate a received block and add it to the ledger

For the time being, the query is generated manually. We defined it externally to analyze a global property that relied on local properties. However, any query can be generated by a node or an external entity as long as it follows our packet definition. The transaction is generated when the node receives a query, then analyzes the query and the logs and if there is a result, it assembles a transaction and sends it. On the other side, if this module receives a transaction, it checks the query it belonged and based on that it determines if there is enough information to start the consensus algorithm. Finally, when the node receives a block, it validates it by calculating a hash of specific data. The block id represents a hash of the previous block id, a nonce, a timestamp, the IP of the block assembler or miner, the result and a Merkle tree of the transactions. By doing this, we can remove the transactions of the block. This is because, when a transaction is generated, it is broadcasted to the network. Therefore if we look at the network a moment before the block is created, all nodes in the network have all transactions that have been generated. Due to this, it is not necessary to resend the transactions in the block packet. Then every node that receives a block does an integrity validation to make sure that the information received is correct. This is done by a simple computation of the block id based on the transactions and the information of the block. It is worth mentioning that for the three main tasks of this module, the packets sent and received are handled by the router module. This means that this module focuses only on the three main tasks and does not handle any broadcasting or any other activities to ensure the reception of a packet, this is done by the router module.

5.4.2 Miner module

The miner module takes care of two simple tasks which are:

a) Start the consensus

b) Create the block if elected leader

The ledger module triggers an internal packet to notify that there is enough information to aggregate a query. The criteria to decide if there is enough information or not, is given by the query parameters. When this internal packet is received, the miner module triggers raft and then waits for a response of this module. If the raft module acknowledges that itself is the leader, it continues to calculate the result of the local views on the global view if that is the case. Then it assembles the block and sends it. It is worth mentioning that for the two tasks of this module, the packets sent and received (including the internal packets) are handled by the router module.

5.4.3 Raft module

The raft module handles the leader election process of the consensus algorithm. This module is the most critical part of the consensus. It follows the algorithm described in Section 4.2.2. It is worth mentioning that, for this module to fully determine that a node is a leader, it has to accumulate seven pings from the leader. Assuming that a leader is elected, it starts sending pings, and if he achieves seven consecutive pings, this determines that this node is the leader. This is done to assure that there are not multiple leaders at the same time. The number of pings we used was determined arbitrarily to give some time to ensure that the results are not affected by the fragmentation. Once a node has accumulated this ping threshold, it notifies the miner to end the consensus. For this module, the message exchange to determine the leader as the request for votes, vote and ping are done directly. This means that this module sends packets from raft module to another raft module without intermediaries. The only packets that are sent through the router module are the start raft, and end raft packets, which tells the raft module to start or the raft module acknowledges that there is a leader elected.

5.4.4 Router module

The router module takes care of the message interchange between the wireless medium and the other modules. When other modules send a message to the router, this takes care of sending it. It keeps track of the messages sent. Therefore it re-broadcasts a message when a message does not receive an acknowledgment. It also makes sure that, when it re-broadcasts a message coming from another node, it is only rebroadcasted one time to avoid unnecessary noise in the medium.

5.5 Summary

The emulator herein proposed acted as a pillar for our experiments shown in the previous sections. The different stages of our emulator are directly correlated to the number of emulations that we were running. In Section 3.3.2 we ran 250 emulations, in Section 3.3.3 we ran 1,000, in Section 3.3.4 we ran 10,000, and finally in Section 4.3 we ran 76,800 emulations. The increments in the number of emulations shown the efficiency of the stages of our emulator. The first stage of the emulator is based on NS-3, Docker, and the Linux Kernel. Then, we decided to make it scalable by using Amazon Web Services. Finally, we supported the scalability by adding a data processing mechanism to analyze more efficient the overall results. Alongside the creation, fellow researchers started using the emulator which comes with great excitement from our side. We started motivated by complying with the recommendations from the simulations literature. We achieved it for our experiments, and we hope that it helps other researchers to make the same.

On the other side, the implementations although challenging, were somehow straightforward to implement given the strong mathematical background. In overall, we relied on the Go language, which was proposed by Google and in combination with Docker, provides great performance.

6

Conclusion and Future Work

In this chapter, we present some remarks to conclude the work in this thesis and some possible directions for future research. In Chapter 2, we provide an updated state of the art for monitoring in MANETs. We focused on survey the decentralized and distributed literature. In the case of the decentralized monitoring literature, we expanded on gossip-based, hierarchical-based and hybrid-based approaches. For the distributed monitoring literature, we analyzed some proposals but also looked at blockchain-related approaches in the network literature. Through this chapter, it can be seen the many open questions that remain in the literature. We believe that through out this work, it can be observed that numerous challenges were tackled.

To facilitate the reading, the future work arising from the current studies will be *emphasized*.

In Chapter 3, we presented a hybrid-based approach for monitoring MANETs in a decentralized fashion. It consists of the combination of gossip-based and hierarchical-based approaches. It tries to maximize the advantages of both approaches while minimizing the drawbacks of them to provide a robust and scalable solution for more extensive scenarios. The approach relies on two procedures: query and aggregate. The query procedure is the process to disseminate a query to all the nodes. This is done relying on an epidemic algorithm to achieve efficient dissemination. Then, once the query is propagated to all the network, the nodes change to the aggregate procedure. In this stage, the approach relies mostly on a hierarchical topology, built from the previous procedure. It is important to highlight that to minimize the frangibility of the topology, it is valid only during this monitoring process. This means that for every monitoring process a new virtual hierarchical topology is derived to avoid maintaining this topology when it is not used. At the same time, it maximizes the aggregation efficiency of the structure. Given that, due to mobility, the topology can physically break, there is a gossip routing fallback mechanism to support the hierarchy. Therefore, if a node cannot find a route to its parent

node, it relies on the gossip mechanism to send the packet accordingly. In this case, we maximize the flexibility of the gossip algorithm by using it to find a route but minimizing the convergence time impact by using only when the topology has lost its physical locality. Although the many novel contributions, we believe that *there is a need to evaluate and experiment on hybrid-approaches with different combinations, so much for the gossip as for the hierarchical component of our approach.* This means that, although we relied on a simple gossiping algorithm, therefore a more specific and robust gossiping algorithm can be used instead. The same goes for the virtual hierarchical topology. We could rely on a different mechanism to derive this topology which could be more efficient.

Based on the results from our experiments, we could observe tremendous and promising results. The accuracy of the approach ranged from ≈70% to practically ≈100%. On the other hand, the convergence time, varied a little bit more since we relied on different timeout values. However, we can conclude that relying on a low timeout, the convergence time was very good. From the results, it can be observed that there is a need for a better timeout. Given that the timeout is the primary mechanism to switch between the query and the aggregate procedure. Therefore, there is room for *include a different mechanism to switch between procedures or to minimize this value to an optimal value.* Another possibility could also be to *include a learning algorithm to determine the timeout more efficiently.* It is essential to understand that if the overall convergence time, which for the time being is highly dependent on the timeout, can be decreased, it would mean that the virtual hierarchical topology would exist for a shorter period. If this is achieved, it will imply that the frangibility of the topology would be minimized. Therefore it would result in a more accurate and efficient overall approach.

Based on our study, we concluded the following:

(i) The requirement to provide more mathematical structures to define novel approaches, we achieved this by defining an automaton describing the protocol. Which provides an efficient way of representing a protocol and a mathematical test ground. It also facilitates the implementation of the approach, and it could be used as support to be implemented in other languages that we did not cover. This is critical background support which *enables the opportunity to perform model checking to the automata presented*. This could derive exciting results for improvements and guarantees of the model herein presented.

(ii) The recommendation of a monitoring packet definition relying on a definition using a well-known data representation language, like the case of JSON (Bray, 2014). This packet can ease the communications and the implementations to rely on this approach. Although it could be argued that a packet defined in a byte structure could be more efficient, we intend to find a middle-ground where easiness and usability are not compromised. *There is an opportunity to evaluate the efficiency and usability of other data representation languages.* Although, to the best of our knowledge, JSON is the de facto for many machine-to-machine communications, especially for the web APIs.

(iii) The importance and the versatility of a novel decentralized monitoring architecture through the combination of a monitoring protocol and a hybrid-based approach. To the best of our knowledge, there are not many proposals in this hybrid category, and based on the results we can conclude that the results are promising. The approach could also provide valuable insight

into wired scenarios or any MANET-derived scenarios. Therefore, *there is an opportunity to evaluate the feasibility of this approach into other types of networks.*

For our approach, we iterated successfully on the improving of the accuracy by introducing the concept of multi-root node approach. Through this improvement, the network coverage is enhanced, the impact of the network fragmentation is reduced, but the number of network traffic is increased. This allowed us to achieve an overall of ≈10% improvement to the accuracy of our approach, ranging from ≈80% to practically ≈100%. Throughout the experiments, we relied on two random nodes to be the root nodes. This raises the question if another parameter is added (e.g., energy consumption or location) could the overall approach be improved. For this, we believe that *it would be interesting to introduce a new mechanism of root nodes selection, for instance, it could be based on location, energy, computing power or other parameters.* For the time being, the multi-root nodes run the monitoring process through all the nodes in the network. We think that there could be an improvement by doing a *partial monitoring by different root nodes and then converging the results among them*, trying to reduce the network traffic. Finally, we can conclude that the decentralized approach proposes exciting answers to some of the open questions but at the same time drove new exciting research directions and challenges.

In Chapter 4, we have presented a truly distributed architecture for monitoring mobile ad-hoc networks through a combination of a distributed ledger, consensus algorithm, and a network protocol. The proposal of this idea is from the Blockchain technology, which we can conclude that it could bring a lot of exciting proposals but it is not a solution that fits all problems. Through this idea, we provided a robust and scalable solution, taking advantage of the best qualities for the used technologies focused on the MANET context. All of this was done by first providing a mathematical background, which allowed us to elaborate mathematically, the definitions, dependencies and results for the local views and most important for the global views of the network. We realized the need for these definitions from our previous works, and we decided that this would bring an interesting bedrock to our proposal. The architecture starts by propagating a query that contains a global property and a set of local properties to satisfy. Each node calculates the local properties, and if it can satisfy it, it replies a transaction to the network. Then based on the query criteria, all the nodes can switch to aggregate and complete the query. A consensus algorithm does this, and this allows the network to choose a leader for that query in a distributed, deterministic and democratized way. This node aggregates the local views and computes the global property. The process derives a block as a result, which is sent to the network and all the nodes agree on the result. We inspired our consensus algorithm in a protocol called Raft. This protocol is quite popular due to its efficiency. Most of the consensus algorithms literature is designed for non-mobile networks, and most of the industrial interest is for data centers which are not mobile at all. Therefore, *we think that there is a need to analyze and evaluate a consensus algorithm designed for wireless networks.* It is important to highlight that each node can validate, audit and agree on a result. Which not only allows the network to consider security concerns but also to ensure the trustability of the data, and all of this is done by design. Based on our study, we concluded that the approach has an incredibly high accuracy compared to other approaches.

Nonetheless, given the traction that the blockchain has gained in the recent years, it is fair to assume that this might provide the necessary push to study a broad set of consensus algorithms,

amongst them the mobile supported consensus algorithms. The blockchain technology could also provide some interest in testing slim or minified version of blockchain for specific scenarios like ours. Recently, blockchain has been a massive topic of debate, and this could positively impact many computer science areas. The results we obtained from our definitions and experiments are a proof of the versatility of the blockchain technology when its adequately implemented. It is not that other blockchain implementations are wrong, but for many cases, the tendency is to include far more complex architectures to simple problems. Finally, the overall architecture sets an interesting result relying on novel technologies and exciting accurate results.

Based on our study, we concluded the following:

(i) The strong mathematical background describing candidate events, local properties, global properties and all the following structures for our architecture allowed us to work on top of a more confident mathematical background. This provides a robust mathematical test ground for the theory. It also provides an excellent resource for the implementations, which can follow the definitions and algorithms to achieve an accurate implementation. The importance of this definition is in the support it provides to model complex and distributed behaviors, e.g., the integrity of the communications. Moreover, that the definitions allow this behavior to be analyzed autonomously by the network itself without the need for central entities nor any intermediary. Although our primary focus is on communication protocols, we firmly believe that this work could be useful in other protocols or network properties. Therefore, we believe that there is *an opportunity to define more sets of messages or a more broad set of messages to model another type of properties and behaviors.*

(ii) The importance and the accuracy of a novel distributed monitoring architecture through the combination of a distributed ledger and a consensus algorithm. It is worth highlight that this was proven by the emulation of multiple scenarios, combining different parameters, and ran multiple times. Alongside this, it followed the best scientific recommendations like providing the code repositories for future testing. This was done following as an inspiration and background studies that define good practices for network simulations (Kurkowski et al., 2005). Although there were numerous experiments and results, we think that there is always room for more experiments. Therefore *it would be interesting to run experiments with multiple queries at the same time while varying more the node densities and node speed.* It could also drive an interesting discussion *the observation in the experiments of other properties in the MANET, i.e., power consumption and trustability.* Another interesting discussion could be derived by *analyzing the experiments from another perspective, i.e., security.* In the end, we can conclude that our experiments provide a promising foundation for future works.

In Chapter 5, we outline the motivation to build our emulator. Which came from the necessity to perform an accurate, repeatable and vast amount of experiments. This motivation came from reviewing the network simulation literature and finding consistently studies stating that there is a lack of credibility for network proposals. Therefore we designed a scalable and configurable testbed using NS3 and Docker that illustrates the effectiveness of our approaches for a different number of nodes, speeds and mobility patterns. We started with small iterations, and as we conducted more experiments, we also

improved the emulator. This allowed us to conclude into three different stages of our emulator. The first stage was the automated orchestration between NS3 and Docker through different bash scripting support. Then the second stage was the scalability of our approach by building on top an Amazon Web Services orchestrator. Finally, the third stage was the provision of analysis tools to efficiently process the massive amounts of data generated by the experiments. From a subjective point of view, we thought we had developed a traditional tool that allowed us to run a great campaign of experiments. However, then, through statistics provided by Github and email communications, we realized that other researchers were using our approach. It was not only being used, but the people using it were indeed devoted to understanding it. We determined this due to the questions they were doing in email communications. This took us to the conclusion that existing network simulators are useful, but researchers are looking for novel approaches to test their proposals in more real-like simulations. This was a side effect that we did not anticipate, but once again, led us to the conclusion that network emulators are needed.

The effectiveness of the emulator can be seen reflected in the numerous experiments through this thesis. The results we achieved showed to some extent the importance of our emulator. It is important to mention that the emulator was not only useful regarding generating results, but it was also effective regarding time. Through the many iterations, we achieved a very high experiments-to-results ratio that allowed us to inspect more in-depth our results. The emulator provides a strong foundation for making our scientific experiments repeatable, reliable and credible. We firmly believe that it could help many other researchers to achieve the same and in overall promote better credibility of scientific experiments in the context of networks. Alongside, some interesting challenges arose from this work. Docker, which is one of the critical components of our emulator is a new but popular container solution. Therefore, some internal functioning is still broadly or not documented, this limited us to achieve a certain level of integration between Docker and NS-3. We firmly believe that *there is a need for a stronger integration between our emulator and Docker*, to create a more dynamic network configuration that allows a broader range of experiments that can be run in it. We think that due to the many efforts in the NS-3 and Docker software development cycles, *there is a need to provide a more streamed line to keep both versions up to date to provide a full range of capabilities of both software*. Regardless of the versatility of our emulator, due to NS-3 software limitations, it can not simulate our desired behavior for a large number of nodes. Due to the singularity of the case, *it remained an open issue on how to improve NS-3 simulations for a large number of nodes.* Moreover, general testing and information about the emulator are needed. For the time being, we have these empirical assumptions on the CPU and memory consumption, which we used to keep the simulations running under a feasible threshold. Nonetheless, there is no information or statistics about the actual resource consumption usage and actual limitations. Therefore we can conclude that *there is a need to measure the resource impact and limitations of the emulator.* Finally, we can conclude in overall, that the emulator provides an incredible bedrock for experiments allowing us to not only run massive campaigns of experiments but to also with a high level of credibility based on the recommendations of the literature.

Bibliography

NS-3. (2010). Howto use linux containers to set up virtual networks. Retrieved from https://www.nsnam.org/wiki/HOWTO_Use_Linux_Containers_to_set_up_virtual_networks. (Cited on page 93)

Abid, S. A., Othman, M., & Shah, N. (2014). 3d p2p overlay over manets. *Computer Networks*, *64*, 89–111. (Cited on page 55).

Aspnes, J. & Ruppert, E. (2009). An introduction to population protocols. In *Middleware for network eccentric and mobile applications* (Pages 97–120). Springer. (Cited on page 12).

Bari, M. F., Boutaba, R., Esteves, R., Granville, L. Z., Podlesny, M., Rabbani, M. G., . . . Zhani, M. F. (2013). Data center network virtualization: a survey. *IEEE Communications Surveys & Tutorials*, *15*(2), 909–928. (Cited on page 92).

Basagni, S., Conti, M., Giordano, S., & Stojmenovic, I. (2004). *Mobile ad hoc networking*. John Wiley & Sons. (Cited on page 1).

Battat, N. & Kheddouci, H. (2011). Hman: hierarchical monitoring for ad hoc network. In *Embedded and ubiquitous computing (euc), 2011 ifip 9th international conference on* (Pages 414–419). IEEE. (Cited on page 15).

Battat, N., Makhoul, A., Kheddouci, H., Medjahed, S., & Aitouazzoug, N. (2017). Trust based monitoring approach for mobile ad hoc networks. In *International conference on ad-hoc networks and wireless* (Pages 55–62). Springer. (Cited on page 18).

Battat, N., Seba, H., & Kheddouci, H. (2014). Monitoring in mobile ad hoc networks: a survey. *Computer Networks*. (Cited on pages 9, 26, 44 and 55).

Belfkih, A., Duvallet, C., Amanton, L., & Sadeg, B. (2015). A new query processing model for maintaining data temporal consistency in wireless sensor networks. In *Intelligent sensors, sensor networks and information processing (issnip), 2015 ieee tenth international conference on* (Pages 1–6). IEEE. (Cited on page 18).

Belfkih, A., Duvallet, C., Sadeg, B., & Amanton, L. (2017). A real-time query processing system for wsn. In *International conference on ad-hoc networks and wireless* (Pages 307–313). Springer. (Cited on page 18).

Bray, T. (2014). *The javascript object notation (json) data interchange format*. RFC Editor. (Cited on pages 6, 36, 75 and 106).

Burrows, M. (2006). The chubby lock service for loosely-coupled distributed systems. In *Proceedings of the 7th symposium on operating systems design and implementation* (Pages 335–350). USENIX Association. (Cited on page 56).

Calarco, G. & Casoni, M. (2013). On the effectiveness of linux containers for network virtualization. *Simulation Modelling Practice and Theory*, *31*, 169–185. (Cited on pages 93 and 95).

Cavalcanti, E. R., de Souza, J. A. R., Spohn, M. A., Gomes, R. C. d. M., & Costa, A. F. B. F. d. (2018). Vanets' research over the past decade: overview, credibility, and trends. *ACM SIGCOMM Computer Communication Review*, *48*(2), 31–39. (Cited on page 90).

Chakchouk, N. (2015). A survey on opportunistic routing in wireless communication networks. *IEEE Communications Surveys & Tutorials*, *17*(4), 2214–2241. (Cited on pages 11 and 26).

Chan, M.-C., Chen, C., Huang, J.-X., Kuo, T., Yen, L.-H., & Tseng, C.-C. (2014). Opennet: a simulator for software-defined wireless local area network. In *Wireless communications and networking conference (wcnc), 2014 ieee* (Pages 3332–3336). IEEE. (Cited on page 93).

Che, X., Maag, S., Nguyen, H. N., & Zaïdi, F. (2015). Guiding testers' hands in monitoring tools: application of testing approaches on SIP. In *Testing software and systems - 27th IFIP WG 6.1 international conference, ICTSS 2015, sharjah and dubai, united arab emirates, november 23-25, 2015, proceedings* (Pages 105–123). (Cited on page 58).

Che, X., Maag, S., Tan, H., Tan, H., & Zhou, Z. (2015). A passive testing approach for protocols in wireless sensor networks. *Sensors*, *15*(11), 29250–29272. (Cited on page 55).

Chlamtac, I., Conti, M., & Liu, J. J.-N. (2003). Mobile ad hoc networking: imperatives and challenges. *Ad hoc networks*, *1*(1), 13–64. (Cited on page 1).

Conti, M. & Giordano, S. (2007a). Multihop ad hoc networking: the reality. *IEEE Communications Magazine*, *45*(4). (Cited on page 2).

Conti, M. & Giordano, S. (2007b). Multihop ad hoc networking: the theory. *IEEE Communications Magazine*, *45*(4). (Cited on page 2).

Conti, M. & Giordano, S. (2014). Mobile ad hoc networking: milestones, challenges, and new research directions. *IEEE Communications Magazine*, *52*(1), 85–96. (Cited on pages 1, 3 and 10).

Cormode, G. (2011). Continuous distributed monitoring: a short survey. In *Proceedings of the first international workshop on algorithms and models for distributed event processing* (Pages 1–10). ACM. (Cited on page 55).

Cormode, G. (2013a). The continuous distributed monitoring model. *ACM SIGMOD Record*. (Cited on page 3).

Cormode, G. (2013b). The continuous distributed monitoring model. *ACM SIGMOD Record*. (Cited on pages 24, 25 and 26).

Dam, M. & Stadler, R. (2005). A generic protocol for network state aggregation. *self*, *3*, 411. (Cited on page 15).

Dolev, S., Israeli, A., & Moran, S. (1993). Self-stabilization of dynamic systems assuming only read/write atomicity. *Distributed Computing*, *7*(1), 3–16. (Cited on page 15).

Drabkin, V., Friedman, R., Kliot, G., & Segal, M. (2007). Rapid: reliable probabilistic dissemination in wireless ad-hoc networks. In *Reliable distributed systems, 2007. srds 2007. 26th ieee international symposium on* (Pages 13–22). IEEE. (Cited on pages 30, 83 and 84).

Fischer, M. J. (1983). The consensus problem in unreliable distributed systems (a brief survey). In *International conference on fundamentals of computation theory* (Pages 127–140). Springer. (Cited on page 56).

Füßler, H., Widmer, J., Käsemann, M., Mauve, M., & Hartenstein, H. (2003). Contention-based forwarding for mobile ad hoc networks. *Ad Hoc Networks*, *1*(4), 351–369. (Cited on page 14).

Goodloe, A. E. & Pike, L. (2010). Monitoring distributed real-time systems: a survey and future directions. (Cited on pages 5, 54 and 56).

Google. (2014). Method and node for finding content in a content distribution network, and method for creating a virtual representation of a content distribution network, patent us 8665757 b2. (Cited on page 29).

Guerrieri, A., Carreras, I., De Pellegrini, F., Miorandi, D., & Montresor, A. (2010). Distributed estimation of global parameters in delay-tolerant networks. *Computer Communications*, *33*(13), 1472–1482. (Cited on page 12).

Handigol, N., Heller, B., Jeyakumar, V., Lantz, B., & McKeown, N. (2012). Reproducible network experiments using container-based emulation. In *Proceedings of the 8th international conference on emerging networking experiments and technologies* (Pages 253–264). ACM. (Cited on page 93).

Huang, L. (2005). Virgo: virtual hierarchical overlay network for scalable grid computing. In *European grid conference*. Springer. (Cited on page 29).

Jacquet, P., Muhlethaler, P., Clausen, T., Laouiti, A., Qayyum, A., & Viennot, L. (2001). Optimized link state routing protocol for ad hoc networks. In *Multi topic conference, 2001. ieee inmic 2001. technology for the 21st century. proceedings. ieee international* (Pages 62–68). IEEE. (Cited on page 44).

Jelasity, M., Montresor, A., & Babaoglu, O. (2005). Gossip-based aggregation in large dynamic networks. *ACM Transactions on Computer Systems (TOCS)*, *23*(3), 219–252. (Cited on page 12).

Kempe, D., Dobra, A., & Gehrke, J. (2003). Gossip-based computation of aggregate information. In *Foundations of computer science, 2003. proceedings. 44th annual ieee symposium on* (Pages 482–491). IEEE. (Cited on page 13).

Khan, M. A., Peters, S., Sahinel, D., Pozo-Pardo, F. D., & Dang, X.-T. (2018). Understanding autonomic network management: a look into the past, a solution for the future. *Computer Communications*, *122*, 93–117. (Cited on page 3).

Kiess, W. & Mauve, M. (2007). A survey on real-world implementations of mobile ad-hoc networks. *Ad Hoc Networks*, *5*(3), 324–339. (Cited on page 1).

Krishnamurthy, S., Ardelius, J., Aurell, E., Dam, M., Stadler, R., & Wuhib, F. (2010). The accuracy of tree-based counting in dynamic networks. *arXiv preprint arXiv:1004.4559*. (Cited on page 15).

Kurkowski, S., Camp, T., & Colagrosso, M. (2005). Manet simulation studies: the incredibles. *ACM SIGMOBILE Mobile Computing and Communications Review*, *9*(4), 50–61. (Cited on pages 90, 91 and 108).

Lalanne, F. & Maag, S. (2013). A formal data-centric approach for passive testing of communication protocols. *IEEE/ACM Trans. Netw. 21*(3), 788–801. (Cited on page 55).

Lamport, L. et al. (2001). Paxos made simple. *ACM Sigact News*, *32*(4), 18–25. (Cited on page 56).

Lamport, L., Shostak, R., & Pease, M. (1982). The byzantine generals problem. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, *4*(3), 382–401. (Cited on page 56).

Li, T. (2016). *Gradient-Damage Modeling of Dynamic Brittle Fracture: Variational Principles and Numerical Simulations* (Doctoral dissertation, Université Paris-Saclay). (Cited on page iv).

Mouttappa, P., Maag, S., & Cavalli, A. R. (2013). Using passive testing based on symbolic execution and slicing techniques: application to the validation of communication protocols. *Computer Networks*, *57*(15), 2992–3008. (Cited on page 55).

Nakamoto, S. (2008). Bitcoin: a peer-to-peer electronic cash system," http://bitcoin.org/bitcoin.pdf. (Cited on page 57).

Nguyen, H. N., Poizat, P., & Zaïdi, F. (2012). Online verification of value-passing choreographies through property-oriented passive testing. In *14th international IEEE symposium on high-assurance systems engineering, HASE 2012, omaha, ne, usa, october 25-27, 2012* (Pages 106–113). (Cited on page 58).

Ongaro, D. & Ousterhout, J. K. (2014). In search of an understandable consensus algorithm. In *Usenix annual technical conference* (Pages 305–319). (Cited on pages 56 and 73).

Pawlikowski, K., Jeong, H.-D., & Lee, J.-S. (2002). On credibility of simulation studies of telecommunication networks. *IEEE Communications magazine*, *40*(1), 132–139. (Cited on page 90).

Qin, J., Ma, Q., Shi, Y., & Wang, L. (2017). Recent advances in consensus of multi-agent systems: a brief survey. *IEEE Transactions on Industrial Electronics*, *64*(6), 4972–4983. (Cited on page 56).

Raza, N., Aftab, M. U., Akbar, M. Q., Ashraf, O., & Irfan, M. (2016). Mobile ad-hoc networks applications and its challenges. (Cited on page 2).

Sarkar, N. I. & Gutiérrez, J. A. (2014). Revisiting the issue of the credibility of simulation studies in telecommunication networks: highlighting the results of a comprehensive survey of ieee publications. *IEEE Communications Magazine*, *52*(5), 218–224. (Cited on page 90).

Selimi, M., Kabbinale, A. R., Ali, A., Navarro, L., & Sathiaseelan, A. (2018). Towards blockchain-enabled wireless mesh networks. *arXiv preprint arXiv:1804.00561*. (Cited on page 19).

Sharfman, I., Schuster, A., & Keren, D. (2007). A geometric approach to monitoring threshold functions over distributed data streams. *ACM Transactions on Database Systems (TODS)*, *32*(4), 23. (Cited on page 55).

Sistla, K., George, A., Todd, R., & Tilak, R. (2000). Performance analysis of flat and layered gossip services for failure detection and consensus in scalable heterogeneous clusters. In *Parallel and distributed processing symposium., proceedings 15th international* (Pages 802–809). IEEE. (Cited on page 17).

Sistla, K., George, A. D., & Todd, R. W. (2003). Experimental analysis of a gossip-based service for scalable, distributed failure detection and consensus. *Cluster Computing*, *6*(3), 237–251. (Cited on page 17).

Stingl, D., Gross, C., Nobach, L., Steinmetz, R., & Hausheer, D. (2013). Blocktree: location-aware decentralized monitoring in mobile ad hoc networks. In *Ieee 38th conf. local computer networks* (Pages 373–381). (Cited on pages 14 and 45).

Stingl, D., Groß, C., & Saller, K. (2013). Decentralized monitoring in peer-to-peer systems. In *Benchmarking peer-to-peer systems* (Pages 81–111). Springer. (Cited on page 11).

Stingl, D., Gross, C., Saller, K., Kaune, S., & Steinmetz, R. (2012). Benchmarking decentralized monitoring mechanisms in peer-to-peer systems. In *3rd acm/spec international conference on performance engineering* (Pages 193–204). (Cited on pages 4, 11, 24 and 26).

Stingl, D., Retz, R., Richerzhagen, B., Gross, C., & Steinmetz, R. (2014). Mobi-g: gossip-based monitoring in manets. In *Ieee network operations and management symposium*. (Cited on pages 11 and 44).

Subramaniyan, R., Raman, P., George, A. D., & Radlinski, M. (2006). Gems: gossip-enabled monitoring service for scalable heterogeneous distributed systems. *Cluster Computing*, *9*(1), 101–120. (Cited on page 16).

To, M. A., Cano, M., & Biba, P. (2015). Dockemu–a network emulation tool. In *Advanced information networking and applications workshops (waina), 2015 ieee 29th international conference on* (Pages 593–598). IEEE. (Cited on pages 91 and 93).

Van De Bovenkamp, R., Kuipers, F., & Van Mieghem, P. (2012). Gossip-based counting in dynamic networks. In *International conference on research in networking*. Springer. (Cited on pages 11 and 44).

Winter, T., Thubert, P., Brandt, A., Hui, J., Kelsey, R., Levis, P., ... Alexander, R. (2012). *Rpl: ipv6 routing protocol for low-power and lossy networks*. (Cited on page 18).

Wuhib, F., Dam, M., Stadler, R., & Clem, A. (2009). Robust monitoring of network-wide aggregates through gossiping. *IEEE Transactions on Network and Service Management*, *6*(2), 95–109. (Cited on page 12).

université PARIS-SACLAY

ÉCOLE DOCTORALE
Sciences mécaniques et énergétiques, matériaux et géosciences (SMEMAG)

Titre : [DisMonTest] Une Méthode de test fonctionnel en-ligne basée sur une approche de monitorage distribuée continue appliquée aux systèmes communicants.

Mots clés : Test en-ligne, systèmes distribués, monitorage continu, monitorage fonctionnel, méthodes formelles.

Résumé : Les réseaux MANET représentent un domaine de recherche important en raison des nombreuses opportunités découlant des problématiques et des applications inhérentes à ce type de réseau. Les problématiques les plus récurrentes sont la mobilité, la disponibilité ainsi que les ressources limitées. Un intérêt bien connu dans les réseaux et donc dans les MANET est de monitorer les propriétés de ce réseau et de ses nœuds. Les contraintes des MANET peuvent avoir un impact significatif sur les efforts mis en œuvre pour les monitorer. La mobilité et la disponibilité peuvent créer des résultats incomplets pour le monitorage. Les propriétés usuelles utilisées en monitorage sont simples, comme notamment la consommation moyenne du processeur, la bande passante moyenne, etc. De plus, l'évolution des réseaux a conduit à un besoin croissant d'examiner des comportements plus complexes, dépendants et imbriqués. La littérature indique que la précision des valeurs obtenues par monitorage et donc des approches n'est pas fiable et difficile à atteindre en raison des propriétés dynamiques du MANET. Nous proposons donc des architectures de surveillance décentralisées et distribuées qui reposent sur des multiples points d'observation. L'approche décentralisée combine des algorithmes dits hiérarchiques et de 'gossip' pour fournir une approche de monitorage efficace. Grâce à des expérimentations approfondies, nous avons conclu que même si nous étions en mesure d'atteindre d'excellentes performances, la fragmentation du réseau a toujours un impact sévère sur la méthodologie mise en place. Pour améliorer notre technique, nous avons proposé une approche distribuée pour augmenter l'efficacité et la précision globale. Elle fournit un mécanisme de consensus qui lui permet d'agréger de nombreux résultats fournis par plusieurs nœuds et fournit un résultat plus significatif et plus précis. Nous soutenons notre proposition avec de nombreuses définitions mathématiques qui modélisent les résultats locaux pour un seul nœud et les résultats globaux pour le réseau. Nos expériences ont été évaluées avec un émulateur construit en interne qui s'appuie sur Amazon Web Services, NS-3, Docker et GoLang avec un nombre variable de nœuds, la taille du réseau, sa densité, la vitesse des noeuds, les algorithmes de mobilité et les délais. Grâce à cet émulateur, nous avons pu analyser plusieurs aspects en fournissant des testbeds reproductibles, documentés et accessibles. Nous avons obtenu des résultats prometteurs pour les deux approches, et surtout pour l'approche distribuée en particulier en ce qui concerne la précision des valeurs obtenues par monitorage.

Title: A novel online functional testing methodology based on a fully distributed continuous monitoring approach applied to communicating systems

Keywords: Online testing, distributed systems, continuous monitoring, functional monitoring, formal methods.

Abstract: MANETs represent a significant area of network research due to the many opportunities derived from the problematics and applications. The most recurring problematics are the mobility, the availability and also the limited resources. A well-known interest in networks and therefore in MANETs is to monitor

properties of the network and nodes. The problematics of the MANETs can have a significant impact on the monitoring efforts. Mobility and availability can create incomplete results for the monitoring. The usual properties discussed in monitoring are simple ones, e.g., average CPU consumption, average bandwidth and so on. Moreover, the evolution of networks has led to an increasing need to examine more complex, dependant and intertwined behaviors. The literature states that accuracy of the approaches is not reliable and difficult to achieve due to the dynamic properties of the MANET. Therefore we propose a decentralized and distributed monitoring architectures that rely on multiple points of observation. The decentralized approach combines gossip and hierarchical algorithms to provide an effective monitoring approach. Through extensive experimentation, we concluded that although we were able to achieve exceptional performance, network fragmentation still has a harsh impact on the approach. Trying to improve our approach, we proposed a distributed approach, relying on stronger bedrock to enhance the overall efficiency and accuracy. It provides a consensus mechanism that allows it to aggregate and provides a more meaningful and accurate result. We support our proposal with numerous mathematical definition that models local results for a single node and global results for the network. Our experiments were evaluated with an emulator built in-house that relies on Amazon Web Services, NS-3, Docker and GoLang with varying number of nodes, network size, network density, speed, mobility algorithms and timeouts. Through this emulator, we were able to analyze multiple aspects of the approaches by providing a repeatable, documented and accessible test beds. We obtained promising results for both approaches, but for the distributed approach, especially regarding accuracy.

Université Paris-Saclay
Espace Technologique / Immeuble Discovery
Route de l'Orme aux Merisiers RD 128 / 91190 Saint-Aubin, France