University of Cyprus

DEPARTMENT OF COMPUTER SCIENCE

Utilizing Mobile Nodes in Wireless Sensor Networks and IoT Networks

Natalie Temene

A dissertation submitted to the University of Cyprus

in partial fulfillment of the requirements

for the degree of Doctor of Philosophy

December, 2022

VALIDATION PAGE

Doctoral Candidate: Natalie Temene

Doctoral Dissertation Title: Utilizing Mobile Nodes in WSNs and IoT Networks

*The present Doctoral Dissertation was submitted in partial fulfillment of the requirements for the Degree of Doctor of Philosophy at the Department of Computer Science and was approved on **December 12 , 2022** by the members of the Examination Committee.*

Examination Committee:

Research Supervisor ______________________
Professor Chryssis Georgiou

Research Supervisor ______________________
Associate Professor Vasos Vassiliou

Committee Chair ______________________
Associate Professor George Pallis

Committee Member ______________________
Professor Anna Philippou

Committee Member ______________________
Professor Sotiris Nikoletseas

Committee Member ______________________
Assistant Professor Chrysostomos Chrysostomou

DECLARATION OF DOCTORAL CANDIDATE

The present Doctoral Dissertation was submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy of the University of Cyprus. It is a product of original work of my own, unless otherwise mentioned through references, notes, or any other statements.

Natalie Temene

. . . .

Περίληψη

Η λειτουργία του Διαδικτύου των Πραγμάτων και των Ασύρματων Δικτύων Αισθητήρων συχνά μπορεί να διακοπεί λόγω σοβαρών προβλημάτων, όπως τη συμφόρηση, την αποσύνδεση διαδρομής, τα σφάλματα κόμβων και τις επιθέσεις ασφαλείας. Αυτά τα ζητήματα προκύπτουν κυρίως από τους περιορισμούς που έχουν αυτά τα δίκτυα όσον αφορά τη μνήμη, την υπολογιστική ισχύ και την ενέργεια. Σημαντικό ερευνητικό ενδιαφέρον για την αντιμετώπιση των προαναφερθέντων ζητημάτων αποτελεί η μέθοδος που χρησιμοποιεί κινητούς κόμβους που μεταφέρονται από κινητά ρομπότ. Η χρήση κινητών οντοτήτων αυξάνει τους πόρους και τη χωρητικότητα του δικτύου, με απώτερο σκοπό την επέκταση της διάρκειας ζωής του δικτύου.

Στην παρούσα διδακτορική διατριβή παρουσιάζουμε αλγοριθμικές λύσεις που έχουν στόχο τη χρήση και επαναχρησιμοποίηση κινητών κόμβων στο δίκτυο για επίλυση διάφορων προβλημάτων, όπως για παράδειγμα τη συμφόρηση και τα σφάλματα κόμβων.

Η πρώτη αλγοριθμική λύση που παρουσιάστηκε αποτελείται από ένα αλγόριθμο ελέγχου συμφόρησης που χρησιμοποιεί κινητούς κόμβους, και περιλαμβάνει δυο παραλλαγές ώστε να μπορεί να βοηθήσει τους υπάρχοντες αλγορίθμους ελέγχου συμφόρησης στην αντιμετώπιση του προβλήματος. Η πρώτη παραλλαγή χρησιμοποιεί κινητούς κόμβους για να δημιουργήσει τοπικά εναλλακτικά μονοπάτια προς τη βάση, ενώ η δεύτερη παραλλαγή χρησιμοποιεί κινητούς κόμβους για να δημιουργήσει ανεξάρτητα μονοπάτια προς τη βάση.

Μια άμεση επέκταση της πρώτης παραλλαγής του αλγορίθμου, περιλαμβάνει την εισαγωγή του όρου «επαναχρησιμοποίηση» για τους κινητούς κόμβους. Οι κινητοί κόμβοι μπορούν να επαναχρησιμοποιηθούν σε περιπτώσεις όπου το τρέχον πρόβλημά έχει επιλυθεί και δεν χρειάζονται πλέον, καθώς και όταν το ενεργειακό τους επίπεδο τους επιτρέπει να

αναλάβουν μια νέα εργασία. Η προσθήκη του ελέγχου της κατανάλωσης ενέργειας των κινητών κόμβων στο δίκτυο παρέχει τη δυνατότητα της αντικατάστασης ενός εξαντλημένου κινητού κόμβου εγκαίρως πριν εμφανιστεί ένα νέο πρόβλημα ή την επαναχρησιμοποίησή του σε μια νέα θέση για την επίλυση ενός νέου προβλήματος που εμφανίστηκε.

Μια ρεαλιστική επέκταση της προηγούμενης εργασίας περιλαμβάνει τη χρήση κινητών μεταφορέων για τη μετακίνηση των κινητών κόμβων στο δίκτυο. Εδώ εξετάζεται η ιδέα της χρήσης κινητών μεταφορέων που θα είναι υπεύθυνοι για τη μετακίνηση των κινητών κόμβων μέσα στο δίκτυο ώστε να τοποθετηθούν στις υπολογιζόμενες για αυτούς θέσεις. Η επέκταση αυτή αποτελείται από δυο προσεγγίσεις, την παραμονή και την επιστροφή, που βασίζεται στην επόμενη ενέργεια του κινητού μεταφορέα. Στην προσέγγιση παραμονής, ο κινητός μεταφορέας μετά από κάθε μετακίνηση ενός κινητού κόμβου, παραμένει εκεί μαζί του έως ότου λάβει διαφορετικές οδηγίες από τη βάση. Ενώ στην προσέγγιση επιστροφής, ο κινητός μεταφορέας μετά από κάθε μετακίνηση που πραγματοποιεί, επιστρέφει στην αρχική του θέση. Αυτός ο αλγόριθμος είναι σε θέση να χρησιμοποιεί την ενέργεια του κινητού κόμβου στο έπακρο χωρίς να λαμβάνει υπόψη τη διαδικασία επιστροφής, η οποία αποτελεί μέρος του κινητού μεταφορέα, ενώ, παράλληλα, ο κινητός μεταφορέας πραγματοποιεί τη μεταφορά του επιτυχώς και αποτελεσματικά.

Για την επέκταση του εύρους των προβλημάτων που αντιμετωπίζονται, μια νέα αλγοριθμική λύση παρουσιάστηκε που εστιάζει στην αντιμετώπιση σφαλμάτων στο δίκτυο. Παρουσιάζεται ένα Πλαίσιο Διαχείρισης Σφαλμάτων που χρησιμοποιεί κινητούς κόμβους και αποτελείται από δυο μηχανισμούς: ένα αποκεντρωμένο μηχανισμό διαχείρισης σφαλμάτων και ένα κεντρικοποιημένο μηχανισμό διαχείρισης σφαλμάτων. Ο πρώτος μηχανισμός χρησιμοποιεί μια αποκεντρωμένη μέθοδο ανίχνευσης σφαλμάτων και μια τοπική μέθοδο ανάκτησης που χρησιμοποιεί κινητούς κόμβους, ενώ ο δεύτερος μηχανισμός χρησιμοποιεί μια κεντρικοποιημένη μέθοδο ανίχνευσης και ένα μια μέθοδο ανάκτησης που δημιουργεί μονοπάτια που αποτελούνται αποκλειστικά από κινητούς κόμβους.

Οι αλγοριθμικές λύσεις που εστιάζουν στο πρόβλημα της συμφόρησης, δείχνουν σημαντική συμβολή στην άμβλυνση του προβλήματος αυτού τόσο στα Ασύρματα Δίκτυα Αισθητήρων όσο και στο Διαδίκτυο των Πραγμάτων. Επιπλέον, οι αλγοριθμικές λύσεις που εστιάζουν στο πρόβλημα των σφαλμάτων, παρουσιάζουν σημαντική συμβολή στη διαδικασία ανίχνευσης και ανάκτησης σφαλμάτων σε αυτά τα δίκτυα. Όλες οι αλγοριθμικές λύσεις έχουν την προοπτική να χρησιμοποιηθούν για άλλους τύπους δικτύων, καθώς και άλλα προβλήματα που μπορούν να προκύψουν στα δίκτυα αυτά.

Abstract

The operation of the Internet of Things (IoT) networks and Wireless Sensor Networks (WSNs) is often disrupted by a number of problems, such as congestion, path disconnections, node faults, and security attacks. These issues are the results of the limitations these networks provide in terms of memory, computational power, and energy. A method that gains significant research interest for tackling the aforementioned issues is the employment of mobile nodes or nodes deployed by mobile robots. The use of mobile elements essentially increases the resources and the capacity of the network, which results in increasing the lifetime of the network.

This thesis proposes algorithmic solutions that utilize mobile nodes and contribute significantly to the alleviation of different network problems, such as congestion and node failure.

The first algorithmic solution presented is the Node Placement Algorithm (NPA) that consists of two variations to assist existing congestion control algorithms in facing congestion in WSNs. The first variation, called Dynamic Node Placement Algorithm, employs mobile nodes that create locally significant alternative paths leading to the sink, whereas the second variation, called Direct Path Node Placement Algorithm, employs mobile nodes that create individual (disjoint) paths to the sink.

An immediate extension of the NPA algorithm, called Energy-aware Node Placement Algorithm (eNPA), includes the introduction of the term "reuse" to the mobile nodes. Mobile nodes are able to be reused in cases where their current problem has been resolved and they are not needed anymore, as well as when their energy level allows them to take on a new task. The addition of considering the energy consumption of the mobile nodes in the network provided a helpful task. This task was able to replace an energy exhausted mobile node in

time before a new problem occurred or reuse it in a new position for resolving a new problem occurrence.

A realistic extension of the previous work, called Carrier-based Node Placement Algorithm (cNPA), includes the use of mobile carriers that carry the mobile node in the network. The concept of mobile carriers as the transportation means that will move the mobile nodes around the network to position them to the calculated location is examined. This concept was examined via two approaches, stay and leave, based on the next action of the carrier. In the stay approach, the mobile carrier, after each transportation, stays with its assigned mobile node until instructed otherwise from the sink node. In the leave approach, the mobile carrier, after each task, returns to its initial position. This algorithm uses the energy consumption of each mobile node to the fullest without considering the returning process, which is part of the carrier, while the carrier performs its transportation effectively and efficiently.

To extend the range of use cases, a new algorithmic solution is presented that focuses in dealing with faults in the network. The Fault-Tolerant Node Placement Algorithms (FTNPA) is presented that consists of two variations: the decentralized and the centralized. The Decentralized FTNPA uses a decentralized detection mechanism and a local recovery mechanism that uses mobile nodes, whereas the Centralized FTNPA uses a centralized detection mechanism and its recovery mechanism creates alternative mobile nodes paths.

The evaluation of all algorithms was performed with the use of the COOJA simulator of Contiki OS. The different approaches were compared to different scenarios of the algorithm. The results show that all algorithms are able to mitigate the problem that occurs in the network.

The algorithmic solutions that focus on the use case of congestion, show a significant contribution to the alleviation of the problem of congestion in IoT and WSNs. Additionally, the algorithmic solutions that focus on the use case of faults, show a significant contribution

to the detection and recovery process of faults in IoT and WSNs. All algorithmic solutions have the potential of being used for other types of networks, such as Smart Cities and Industrial IoT (IIoT), as well as different network problems, such as malicious attacks and data modifications.

Overall, the thesis combines theory and practice by, on the one hand, developing novel algorithmic solutions addressing challenges at different layers and aspects (such as positioning, energy, transportation, and fault-tolerance), while, on the other hand, implementing and evaluating the solutions using simulators and identifying interesting performance trade-offs for different methods and approaches.

Acknowledgments

My PhD work wouldn't have been possible without the guidance of my thesis supervisors, Professor Chryssis Georgiou and Associate Professor Vasos Vassiliou. Their continuous support, inspiring guidance, invaluable encouragement, and immense knowledge pushed me to sharpen my way of thinking and brought my work to a higher standard. I would like to thank each one of them separately for all the practical and financial support, for keeping their office door always open, and particularly, for the precious time that they invested in me.

My deep gratitude goes towards Dr. Charalampos Sergiou and Dr. Christiana Ioannou for their guidance and persistent help during my academic years. Firstly, I would like to thank Charalampos for his continuous support all these years in my research, as well as, his patience in reviewing my writings. Furthermore, I would like to thank Christiana for always believing in me. Her emotional and practical support during these years was an important guidance for accomplishing this work.

I am very thankful to the team members of the "Foundations of Computing Systems and Theoretical Computer Science Laboratory" and the "Networks Research Laboratory" at the University of Cyprus for being around at times of very intense effort, expressing their support, and providing useful feedback. I am thankful to the members of the faculty of the department of Computer Science with whom I have collaborated over these years as part of my teaching assistance duties, and for always displaying a constructive high standard of professionalism in their duties. Similarly, I would also like to thank the Department's staff who willingly and patiently provided their important services whenever required.

Finally, my deepest gratitude goes to my parents, Kyriacos and Maria, for their care and patience over these long years of my studies. Without their encouragement this would have been impossible. Their support and love gave me the extra push and incentive I needed to remain focused on my goal. I cannot forget to thank my friends, for providing happy distractions to rest my mind outside of my research.

Thesis Contributions

This thesis is founded on the knowledge acquired by the author's involvement in the authorship of the following journal articles and conference papers:

Journal Articles

1. Natalie Temene, Charalampos Sergiou, Chryssis Georgiou, and Vasos Vassiliou. A Survey on Mobility in Wireless Sensor Networks. Ad Hoc Networks, Vol. 125, February 2022.

2. Natalie Temene, Charalampos Sergiou, Christiana Ioannou, Chryssis Georgiou, and Vasos Vassiliou, A Node Placement Algorithm Utilizing Mobile Nodes in WSN and IoT Networks. In Telecom, Vol. 3, No. 1, pp. 17-51, Multidisciplinary Digital Publishing Institute, January 2022.

Conference and Workshop Proceedings

3. Natalie Temene, Andreas Naoum, Charalampos Sergiou, Chryssis Georgiou, and Vasos Vassiliou. A Decentralized Node Placement Algorithm for WSNs and IoT Networks. Proceedings of the 8th IEEE World Forum on Internet of Things, IEEE WFIoT 2022, Yokohama, Japan, 26 October - 11 November, 2022.

4. Natalie Temene, Andreas Naoum, Charalampos Sergiou, Chryssis Georgiou, and Vasos Vassiliou. A Centralized Node Placement Algorithm in WSNs and IoT Networks. Proceedings of the Global Information Infrastructure and Networking Symposium, GIIS 2022, pp. 25-29, Argostoli, Kefalonia Island, Greece, 26 - 28 September, 2022.

5. Natalie Temene, Charalampos Sergiou, Chryssis Georgiou, and Vasos Vassiliou. Utilizing Carriers for the Energy Node Placement Algorithm in WSN and IoT Networks. Proceedings of the 18th International Conference on Distributed Computing in Sensor Systems, DCOSS 2022, pp. 207-214, Los Angeles, California, USA, May 30 - June 1, 2022.

6. Natalie Temene, Charalampos Sergiou, Christiana Ionannou, Chryssis Georgiou, and Vasos Vassiliou. Energy Efficient Mechanism for Reusing Mobile Nodes in WSN and IoT Networks. Proceedings of the 17th International Conference on Distributed Computing in Sensor Systems, DCOSS 2021, pp. 287-294, Online, July 14-16, 2021.

7. Antonia Nicolaou, Natalie Temene, Charalampos Sergiou, Chryssis Georgiou, and Vasos Vassiliou. Utilizing Mobile Nodes for Congestion Control in Wireless Sensor Networks. Proceedings of the 30th IEEE Annual International Symposium on Personal, Indoor and Mobile Radio Communications, PIMRC 2019, pp. 1-7, Istanbul, Turkey, September 8-11, 2019. Also as arXiv preprint arXiv:1903.08989.

8. Antonia Nicolaou, Natalie Temene, Charalampos Sergiou, Chryssis Georgiou, and Vasos Vassiliou. Utilizing Mobile Nodes for Congestion Control in Wireless Sensor Networks. Proceedings of the 15th International Conference on Distributed Computing in Sensor Systems, DCOSS 2019, pp.176-179 Santorini, Greece, May 29-31, 2019.

9. Charalampos Sergiou, Vasos Vassiliou, Chryssis Georgiou, Christiana Ioannou, Natalie Temene, and Aristodemos Paphitis. Competition: Dynamic Alternative Path Selection in Wireless Sensor Networks. Proceedings of the 2017 International Conference on Embedded Wireless Systems and Networks, EWSN 2017, pp. 276-277, Uppsala, Sweden, February 20-22, 2017.

Contents

List of Algorithms

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

Wireless Sensor Networks (WSNs) are a special category of wireless ad-hoc networks that consist of numerous sensor nodes [38,69]. These wireless devices have sensing, computation and communication capabilities. In more detail, the sensor nodes of a network work together to complete a common task, e.g., monitoring a specific area, which is determined by the range of nodes. When an event occurs, the nodes within the sensed area become active and start collecting information. The nodes transmit the data towards the sink(s), where all data from the network are collected. The routing of packets within the network, in a hop-by-hop manner, where nodes send packets to the next hop (node) selected from its neighbouring list, is based on criteria that are normally specified by the application. Such criteria are usually based on the number of hops from sink, the remaining energy, delay, etc.

The main task of WSNs is, in principle, to sense the environment specified by the application, collect data, process them, and finally forward them to the sink(s). The applications [28] of WSNs can be classified into two major categories: monitoring and tracking. Examples of monitoring applications include environmental conditions sensing, hazardous environment exploration, and health monitoring. Tracking applications are found in natural disaster relief operations, tracking animals (usually in wildlife settings), and tracking objects or humans.

In our view, the device and communications part of an Internet of Thins (IoT) system is a direct descendant of WSNs [3, 17], in the sense that it is about networked, resource-constrained systems mainly focusing on low-power wireless devices. As such, many IoT networks exhibit unique characteristics but also come with some important limitations, such as energy, memory and computational power. Energy is a limitation of great importance since the lifetime of the node, and furthermore the network's, depends on it. In particular, the nodes use batteries for energy, which inevitably restrict their lifetime. Normally, due to the power limitations, these nodes suffer from memory and computational power restrictions.

The nature of IoT networks and WSNs is such that it makes them vulnerable to a number of problems, such as path disconnections, node failure, and security attacks. Nowadays, it is important that a network is able to continue functioning correctly even in the occurrence of faults [1]. Another aspect that affects these networks is the increased traffic demand in applications [73]. The hop-by-hop communication method from the sensor nodes to the sink and the limited energy are the main reasons of the problems. As a result, the network may suffer from energy holes or hot-spots, that may frequently result in congestion and network partitioning.

An approach for solving these problems is the use of mobile elements. Mobile elements are able to change their position in the network. Algorithms that use mobile elements normally employ two distinct tactics. The employment of mobile sink(s) or mobile nodes. Mobile sinks have the ability to move around the network and collect data from nodes on the spot. The mobile sink approach mitigates the problem of network disconnection due to energy consumption. On the other hand, the employment of mobile nodes, with similar characteristics as of static nodes, assist existing nodes in performing their tasks, either by replacing energy exhausted or damaged nodes or by creating alternative paths to the sink(s).

Mobility in sensor nodes is able to provide a solution to many challenges that arise in the network. A mobile sensor node that is able to move to different positions can increase the resources and the capacity of the network by being placed in every position that needs assistant. Algorithms that base their operation on mobile nodes are able to improve the lifetime of the network.

1.2 The Extension of the MobileCC Framework

The MobileCC Framework was initially suggested by Koutroullos et al. in [35] and it proposes the concept of utilizing mobile nodes in the network for the creation of alternative paths to the sink. The authors present a mechanism, called Mobile Congestion Control (MobileCC), which was used in certain areas of a network that suffer from congestion repeatedly, permanently or for a long duration. The main idea of the mechanism is to create hard alternative disjoint paths only consisting of mobile nodes to resolve congestion problems that occur in the network. The initial position of the mobile nodes is alongside the sink node. They are in sleep mode and are only moved when notified from the sink to help in a congested area.

This work shows that it is possible to mitigate the effects of congestion by using mobile nodes and create disjoint paths. However, the actual mobile node placement strategy is not addressed.

We now overview the MobileCC Framework. The network consists of randomly deployed static nodes and a set of mobile nodes placed near the sink in sleep mode. The following assumptions are also considered:

- All nodes, both static and mobile, have the same characteristics, such as computation power, communication capabilities, sensing, and transmission range, with the exception of the mobility characteristic of the mobile nodes.
- A simple MAC protocol, such as CSMA/CA, is employed.
- All nodes are aware of their absolute or relative (to the sink) location.
- The sink is informed by the nodes about their location and communication range.

The framework consists of the following mechanisms (see Figure 1.1):

- **Problem Detection Mechanism**: nodes monitor their parameters, such as buffer occupancy or energy consumption, until a certain threshold is reached.
- **Defective Node Selection Mechanism**: the node selects which nodes it will continue serving with the condition that the selected set of sender nodes;s sending rates do not exit a certain threshold.
- **Problem Notification Mechanism**: the defected node informs the sink node about the problem. The message includes all information needed from the neighboring table of the node.
- **Alternative Path Creation Mechanism Using Mobile Nodes**: when the sink is informed about the occurrence of a problem, it will create an alternative path that includes only mobile nodes. The mechanism consists of the following steps:
  - Calculation of Extra Resources
  - Calculation of Optimum Position of Extra Nodes
  - Establishment of Alternative Path

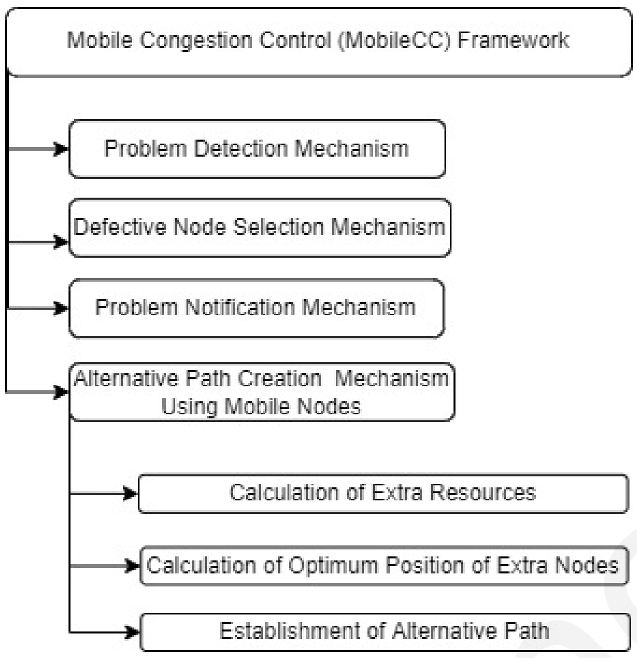

Mobile Congestion Control (MobileCC) Framework
Problem Detection Mechanism
Defective Node Selection Mechanism
Problem Notification Mechanism
Alternative Path Creation Mechanism Using Mobile Nodes
Calculation of Extra Resources
Calculation of Optimum Position of Extra Nodes
Establishment of Alternative Path

Figure 1.1: Diagram of the MobileCC Framework

Concerning the three first mechanisms, a lot of work has already been done so far in the literature. The existing detection mechanisms can be used based on the problem that occurred in the network, such as congestion [70] or failure [54]. However, the initial MobileCC work does not really address the actual mobile node placement strategy. It rather demonstrates, in a before and after fashion, that if mobile nodes are used to create dedicated disjoint paths, it is possible to mitigate the effects of congestion. To address the mobile node placement strategy we designed and implemented different algorithms that are based on calculating the positions of mobile nodes employed in the network. In addition, we focus on faults, with the design of a Mobile Fault-Tolerant Framework for detection and recovery from faults.

1.3 Contribution

This thesis contributes four main novel results by providing algorithmic solutions that utilize mobile nodes to significant problems in WSNs and IoT Networks.

Node Placement Algorithm. The Node Placement Algorithm (NPA) retains the basic principles of MobileCC and addresses the mobile node positioning strategy. It resolves the problem by efficiently and effectively relocating mobile nodes. The main idea is that the Alternative Path Creation mechanism starts when existing congestion control algorithms fail. The algorithm consists of two variations: (1) a dynamic node placement algorithm that solves the problem locally, where the mobile node is placed in such a position that can relieve the affected area, and (2) a direct node placement algorithm that creates a new direct path to the sink, which consists only of mobile nodes.

Energy-aware Node Placement Algorithm. The extended version of the NPA, called the Energy-aware Node Placement Algorithm (eNPA), reuses the mobile relay nodes already in use in the network for resolving congestion, network disconnection, energy holes, or security attack problems occurring in the network. The basic idea of placing mobile nodes in the network to mitigate the problem to be solved is retained, and the focus is set on the energy consumption of the active mobile nodes in the network. Considering the energy levels of a mobile node can be useful in re-using it in a different area of the network or in replacing it on time before causing a new problem in the network.

Carrier-based Node Placement Algorithm. The realistic extension of the eNPA, called Carrier-based Node Placement Algorithm (cNPA), uses mobile carriers to transport the mobile nodes to their destination position. The reason that we choose to employ mobile carriers instead of mobile nodes lies on two facts. The first reason is that mobile nodes impose severe energy costs to the network. The second reason is that networks can accommodate a limited number of these nodes. As a result, it is realistic to state that each sensor node that needs to be replaced in the network is carried by either an Unmanned Ground Vehicle-UGV (i.e. mobile robot) or a Unmanned Aerial Vehicle-UAV (i.e., drone).

Fault-Tolerant Node Placement Algorithm. The use of mobile nodes can be extended to different network problems that can occur in the network, such as faulty nodes. To deal with this problem, we designed the Fault Tolerance Node Placement Algorithm (FTNPA) that uses mobile nodes to recover the network from a faulty state. This algorithm consists of two variations, the decentralized and the centralized, based on the detection method used. The decentralized variation, called Decentralized Fault Tolerance Node Placement Algorithm (DFTNPA), is able to detect a faulty node with the help of the neighboring nodes and recovers the network with a local solution by placing a mobile node in the affected area, whereas the other variation, called Centralized Fault Tolerance Node Placement Algorithm (CFTNPA), uses the sink node to identify the fault and recover the network with the creation of an alternative path of mobile nodes.

All algorithms were evaluated in the COOJA simulator of Contiki OS [58]. The experimental results demonstrate integrity trade-off with respect to percentage of successfully received packets, source to sink delay, packet loss and energy consumption.

This thesis combines theory and practice by, on the one hand, developing novel algorithmic solutions addressing challenges at different layers and aspects, such positioning, energy-awareness, physical transportation of nodes, and fault-tolerance, in an integrated manner, while, on the other hand, implementing and evaluating the solutions using simulators and identifying interesting performance trade-offs for different methods and approaches.

1.4 Document Structure

The remaining parts of this thesis are structured as follows.

In Chapter 2, we present the state of the art for mobility in WSNs and IoT Networks. This chapter presents an overview of the different mobility elements, mobility models and mobility patterns. We focus on protocols that use mobile elements to assist the static nodes in the network. Current research divides these elements into mobile sinks (Sec. 2.2.1) and mobile nodes (Sec. 2.2.2) based on their role in the network and use different solution techniques.

In Chapter 3, we present the Node Placement Algorithm (NPA) that uses mobile nodes to assist the nodes in the network in case of a problem, and consists of two variations, Dynamic Node Placement Algorithm (Sec. 3.1) and Direct Node Placement Algorithm (Sec. 3.2). In Chapter 4, we present the extension of the Dynamic Node Placement Algorithm, called Energy-aware Node Placement Algorithm (eNPA), that reuses the mobile node for new problems that occur in the network. In Chapter 5, we present the realistic extension of the eNPA algorithm, called carrier-based Node Placement Algorithm (cNPA), that uses mobile carriers to transport mobile nodes in the network.

In Chapter 6, we present the experimental evaluation of the three algorithms (NPA,eNPA and cNPA) presented in the previous chapters. Initially, the evaluation setup and the evaluation metrics are given. Then, for each algorithms the evaluation results are shown, as well as a comparison section.

In Chapter 7, we present the Fault-Tolerant Node Placement Algorithm (FTNPA), which uses mobile nodes to assist in case of a failure in the network, and consists of two variations, the Decentralized Fault-Tolerant Node Placement Algorithm - DFTNPA (Sec. 7.2.1) and Centralized Fault-Tolerant Node Placement Algorithm - CFTNPA (Sec. 7.2.2). Also, we mention their experimental evaluation and comparison (Sec. 7.3).

We conclude with Chapter 8, where we overview the thesis work, and discuss future research directions of the presented line of work.

Chapter 2

State of the Art

Mobility in WSNs refers to nodes that have the ability to alter their location after their initial deployment [68]. Algorithms that use mobility can be developed based on two approaches, either with the use of *mobile sink(s)* that receive packets while moving or by using *mobile nodes* that support the task of static nodes. Fig. 2.1 presents the taxonomy of mobility.

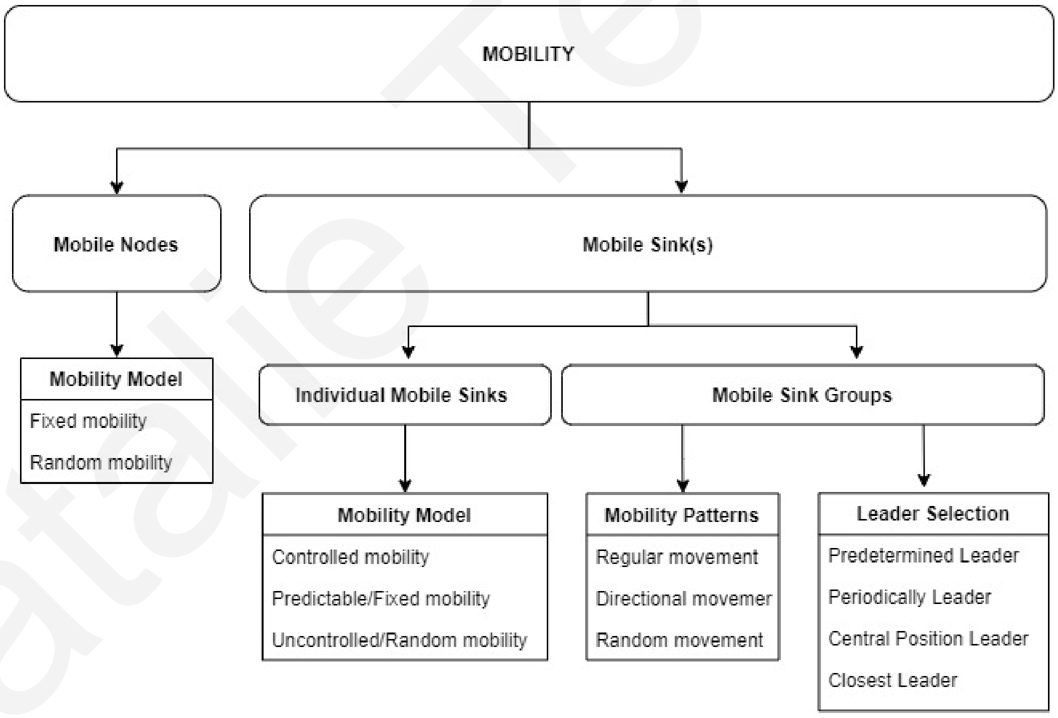
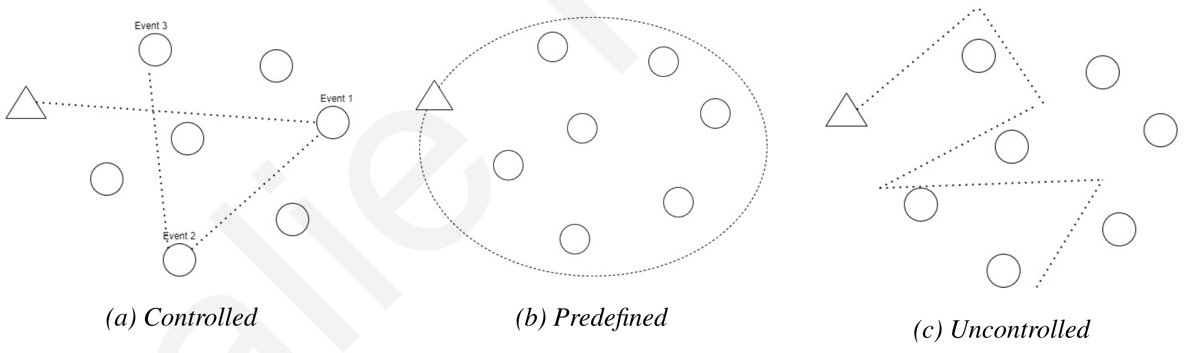


Figure 2.1: Taxonomy of Mobility Solutions

Energy is one of the most challenging constraints of WSNs. Wireless sensor nodes normally use batteries, as their source of energy. Each task executed by a node consumes energy and eventually this will lead into nodes becoming unavailable due to power outage. This means that the lifetime of a node is limited and as a consequence it affects the lifetime of the whole network. To deal with this challenge, recent algorithms aim at maximizing the lifetime of the network by introducing mobility.

The vast majority of existing algorithms in WSNs employ static sinks. A static sink receives the packets from all nodes in the network. The data transmission from node to sink is done in a multi-hop fashion, which frequently creates bottlenecks at the nodes near the sink. Inevitably, these nodes will be depleted and the network will become disconnected. The problem becomes more worrisome in cases where the traffic is constantly routed through the same nodes and as a result only these nodes are depleted. As a result, a possibility of disconnecting the network arises from the creation of holes, while other nodes, residing in different locations, are still fully powered.

The energy consumption among the sensor nodes can be balanced by introducing sink mobility. This approach uses mobile sink(s), that move around the network and collect the packets from the nodes. The mobile sink approach mitigates the problem of network disconnection due to energy consumption. This approach can be classified, based on the behaviour of the mobile sink, as well as on the number of them in the network, into two types, either *individual mobile sinks* or *groups of mobile sinks*.



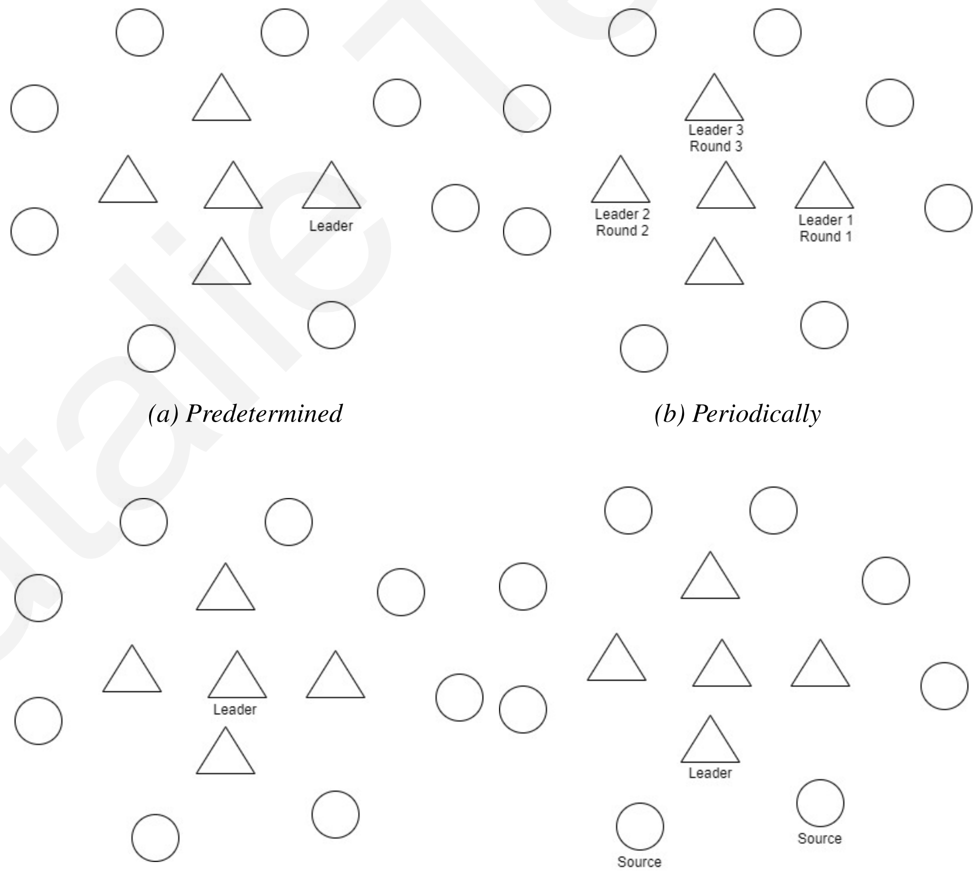
(a) Controlled *(b) Predefined* *(c) Uncontrolled*

Figure 2.2: Mobile Sinks - Mobility Patterns

In the first type, each mobile sink in the network moves individually and independently from the other mobile sinks. In principle, sink mobility in WSNs can follow three basic models. The *controlled mobility model*, *predictable/fixed path mobility model* and the *uncontrolled/random mobility model*. In the controlled mobility model (Fig. 2.2a), the sink follows a controlled or guided path based on some parameters, events or objectives. In the predictable/fixed path mobility model (Fig. 2.2b), a specific path is programmed for the sink to follow in the network with a round robin way. In the uncontrolled/random mobility model (Fig. 2.2c), a random path is followed by the sink with an unpredictable movement in the network that is completely random. The sink is able to request data from its one or more hops distance neighbors.

In addition to these basic patterns, a mobile sink is able to follow a real-life movement pattern. These types of patterns follow a more realistic movement based on real-life examples. These mobile patterns can be divided into: *pedestrian mobility pattern*, *vehicular mobility pattern* and *dynamic mobility pattern*. In the pedestrian mobility pattern, the movement follows the characteristics of walking people that are equipped with sensor nodes attached to their body. In the vehicular mobility pattern, the movement follows the characteristics of vehicles equipped with sensor nodes. In the dynamic mobility pattern, the movement is based on the medium that the node moves through, such as wind or water. Based on the medium type used for the movement, the mobility can be of different dimension (one, two or three dimensional movement).

In the second type, the mobile sinks move as a group. In particular, a group of mobile sinks move closely together around the network and at each region. Each sink can freely move around following the mobility patterns described above, but only inside the specific region.

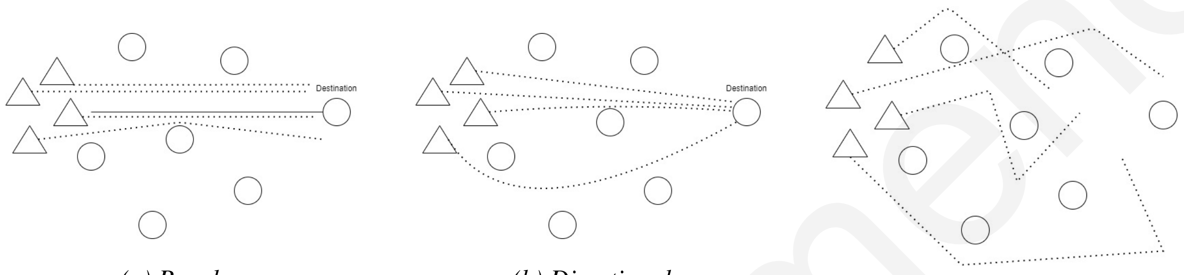


(a) Predetermined *(b) Periodically*

(c) Central *(d) Closest*

Figure 2.3: Mobile Sink Groups - Leader Selection

Each mobile sink group has a leader and the selection of the leader can be performed in four ways. The *predetermined/unchangeable leader* (Fig. 2.3a), where the leader sink will remain until the group of mobile sinks is disbanded. The *periodically changeable leader* (Fig. 2.3b), where the leader sink will change randomly, in a periodically way during each working period of the group. The *central position leader* (Fig. 2.3c), where the leader sink is always the one that is in the center of the group. The *closest leader* (Fig. 2.3d), where the leader sink is the one that is closer to the source node.



(a) Regular *(b) Directional* *(c) Random*

Figure 2.4: Mobile Sink Groups - Mobility Patterns

The *group mobile sinks* type has three mobility patterns, which are based on the patterns from the individual mobile sinks mentioned above. A *regular movement* (Fig. 2.3a), which is the case of the mobile sinks group that move regularly, towards a destination following a straight path. A *directional movement* (Fig. 2.3b), which is the case where the mobile sinks group, move towards a destination, and a *random movement* (Fig. 2.3c), where the mobile sinks group to move around the network in a random pattern, without any destination.

2.1 Review of Existing Surveys on Mobility

In the literature, a limited amount of work exists on surveys about mobility in WSNs, and more specifically the existing surveys talk about mobile sinks and mobile robots.

The protocols mentioned in these surveys are classified based on the different approaches with respect to the mobility. *Table 2.1* shows the classification method of each mobile sink(s)/node(s) survey mentioned in this section.

In 2011, Francesco et al. [14] presented a survey about data collection in WSNs with mobile elements. The main focus of the paper is the communication process between the sensor nodes and the sink node as well as the process of data collection in the network. Initially, the architecture of WSNs with mobile elements is analyzed based on the different components, followed by the description of different types of mobile elements. Furthermore, an

Table 2.1: *The classification methods of each mobile sink/node survey*

Survey	**Category**	**Classification Method**
Francesco et al.(2011) [14]	Mobile Elements	Discovery Data transfer Routing Motion control
Tunca et al. (2014) [81]	Mobile Sinks	Hierarchical Non-hierarchical
Gu et al. (2016) [16]	Sink mobility	UMM-based PRM-based LRM-based URM-based
Latambale et al. (2016) [39]	Sink mobility	mobility patterns
Huang et al. (2019) [22]	Mobile Robots	Collection Delivery Combination
Fissaoui et al. (2020) [11]	Mobile Agents	(itinerary planning) Static Dynamic Hybrid
Singh et al. (2020) [75]	Mobile Elements	(path trajectory) static dynamic
Nguyen et al. (2020) [55]	Mobile sink(s)/node	node mobility sink mobility

overview of the different phases of the data collection process is provided, classifying different approaches. Data collection consists of four phases: *discovery*, *data transfer*, *routing* and *motion collection*. The protocols are classified based on the specific solution approach of each phase.

In 2014, Tunca et al. [81] published a survey about distributed mobile sink routing in WSNs. The survey starts with specifying the advantages and disadvantages of using mobile sinks in WSNs. It continues with presenting different approaches for creating an effective mobile sink routing protocol, like mobility patterns, and concurrently it emphasizes on important design issues that need to be avoided. In the main part of the survey, the protocols are classified into *hierarchical* and *non-hierarchical* routing protocols. The hierarchical class is defined as the most widely adopted approach that aims in decreasing the advertisement of the mobile sink's location, by organizing the network into a virtual hierarchy with each senor imposing a different dynamic role. However, the non-hierarchical routing protocols eliminate the overhead created by the virtual structure and the possibility of hotspots, there is a lack in providing the benefits of the hierarchical approach, like the easy access of the mobile sink's location.

In 2016, Gu et al. [16] presented a survey about sink mobility management in WSNs. The survey at first classifies the mobility models by characterizing the movement pattern of the mobile elements into *uncontrollable model* and *controllable model*. Furthermore, based on the mobility models, the mobility management schemes are classified into four categories: *path-restricted mobility* (PRM), *uncontrollable mobility* (UMM), *unrestricted mobility* (URM) and *location-restricted mobility* (LRM), based on the restriction and behavior of the mobile sink.

Another survey, published in 2016, by Latambale et al. [39] surveys mobile sink techniques in WSNs. The survey starts by classifying the different sink mobility patterns. Three categories are presented, the random/unpredictable mobility pattern, the predictable/fixed-path mobility pattern and controlled mobility pattern. Then, the authors provide a detailed description and comparison of ten different algorithms using mobile sink based on the classification of the moving patterns.

In 2019, Huang et al. [22] presented a survey about mobile robots in WSNs. This survey reviews different techniques related to mobile robots and divide the existing work according to the mobile robots task that is been executed. The task of a mobile robots are: *collection*, *delivery* and *combination*. In the first one, mobile robots are responsible in collecting information from sensor nodes, whereas in the second one, mobile robots are responsible in providing something to the nodes of the network. Mobile robots with a combination task are responsible for collecting and delivering. Each task can be further divided based on the mobility pattern (*random*, *partially controllable* and *fully controllable*) of the robots. The results of this survey show that for the collection task, all three mobility patterns have been part of the existing work. However, in the delivery and combination task, the existing work uses only fully controllable mobility pattern.

In 2020, Fissaoui et al. [11] presented a survey about information fusion using mobile agent itinerary planning in WSNs. A mobile agent is a special software kind that is able to move around the sensor nodes autonomously to achieve its task(s). The mobile agents start from the sink and visit the sensor nodes in their itinerary in order to collect data and return to the sink with the collected data. A mobile agent has two degrees of mobility: *strong* and *weak*, based on the parts used in the transfer process. The itinerary planning for mobile agents is classified into the following three categories: *dynamic*, *static* and *hybrid*, which are further divided into *single* and *multiple*. The dynamic itinerary planning is about the decision of the source to visiting sequence, whereas the static planning is about the selection of the source to visiting set. The hybrid itinerary planning uses both planning for the different method. The authors simulated the most prominent approaches and then evaluated as well as compared them. The implementation process showed that multiple itinerary planning for multiple agents is more complicated than the single itinerary for one mobile agent. The results show that the multiple itinerary planning outperforms the single itinerary planning.

Another survey, published in 2020, by Singh et al. [75] overviews the path planning schemes of mobile elements used in WSNs. A mobile element is a mechanical data carrier that is able to extend the network lifetime, by traveling throughout the network and collected the sensed data from the sensor nodes. Based on the mobility pattern of the mobile elements, their trajectories can be categorized into: *random*, *controlled* and *uncontrolled*. The paper classifies the existing literature into two categories: *static path trajectory* and *dynamic path trajectory*.

Additionally, in 2020, L. Nguyen and H. Nguyen [55] presented a survey about using mobile sink(s) and nodes to effectively maximize the lifetime of a WSN. The lifetime of a WSN is defined as the total amount of time, starting from the initial deployment of the network until the network will not be able to respond to sensing requirement or archive a specific objective. The existing literature is classified based on the type of the mobile element used into: *mobile nodes* and *mobile sink(s)*. The sensor nodes with mobility abilities are able to be used in order to deal with hot-spot problems and effectively reduce the energy consumption of the network. This category can be further divided into: *coverage based strategies*, *connectivity based strategies*, *energy-aware based strategies*, *cooperative computing based strategies*, *localization based strategies* and *clustering based strategies*. The category of mobile sink(s) is further divided into: *one sink mobility for efficient energy* and *multiple sink mobility for efficient energy*. In both categories, the network is able to use either one or multiple mobile sink(s), which are defined based on the mobility pattern used into: *random*, *fixed-path* and *controlled.*

Discussion.

The existing surveys presented above are about a specific category of the mobility research area. Specifically, most of them are about mobile sinks and their technique on collecting the data from the network. Another perspective of these surveys is the use of mobile nodes, which can be called in various names based on their role in the network, like mobile element, mobile agents or mobile robots, and their task of how to move around in the network to collect the data needed. This implies that all of the surveys mentioned focus on the routing technique based on the mobility type of the node.

2.2 Review of Mobility Algorithms

Few algorithms have been proposed in the literature using mobile sink(s)/nodes in WSNs. The research that exists in this topic is limited. These algorithms can be classified based on the type of mobile nodes they employ.

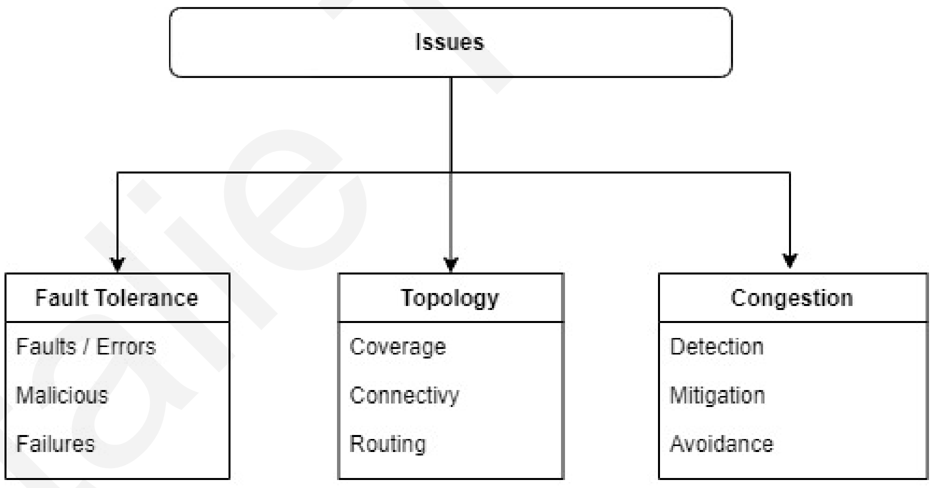


Figure 2.5: Taxonomy of the Issues the Algorithms provide Solution

The algorithms mentioned in this section solve specific problems that appear in the network, see Fig. 2.5. Each classification category has its own problems to resolve, however, some issues are in common. The main idea is the need of maximizing the lifetime of the network and providing an energy-efficient solution. The network disconnection is an issue that can be avoided with the use of nodes that can move. For the mobile sink category, the main problems to be resolved are the data gathering/collection processes. The mobile sink needs to utilize a routing algorithm that will efficiently gather the data from all the nodes in the network. A common solution of this process is to divide the network into clusters [76, 93]. Mobile node algorithms tend to resolve problems like congestion, replacement and coverage. The congestion problem in a network using mobile nodes is efficiently resolved by placing them in the network and providing a new alternative path of the existing nodes. The main goal is to either mitigate or avoid the congestion problem. This solution is also used in solving the hot-spot problem, where the energy is drained faster at the near-sink nodes. Additionally, mobile nodes can be used in a replacing method. This method is used for either an energy exhausted node or a faulty node in the network.

2.2.1 Mobile Sink

As we stated in the previous paragraph, the algorithms that use mobile sink(s), are further divided into algorithms that employ *individual mobile sink* as well as *groups of mobile sink*. The first approach uses mobile sinks that operate independently from the other nodes in the network, whereas the second approach uses mobile sinks that work together as one entity.

Individual Mobile Sink

The algorithms described in this section use a mobile sink that moves independently in respect of the other nodes in the network. Table 2.2 shows the different mobility patterns (Fig. 3.1) of the mobile sink used in the algorithms mentioned below.

Gandham et al. [15] presented the fist attempt on controlled mobile sinks. To increase the lifetime of WSNs, a deployment of multiple and mobile base stations is proposed. Their approach uses an integer linear program (ILP) and a flow-based routing protocol. Mobile sinks work in rounds. During a round, the mobile sinks are stationed and only change their position at the end of the round but before the beginning of the next one. The location of these mobile sinks is determined at the beginning of each round with the use of the ILP model. Afterwards, the flow-based routing protocol is used. The sensor nodes of the network use the flow information, which is obtained by the previous model, to transmit their messages in an energy efficient manner. The algorithm was evaluated using the CPLEX optimizer, a mathematical programming solver for solving difficult problems. The simulation results show that the use of a rigorous approach in order to optimize the energy utilization significantly increases the lifetime of the network.

Liang et al. [42] used the ILP from the previously mentioned approach to provide a mixed integer linear programming (MILP) solution for the problem. The complexity of the

Mobility Pattern	**Algorithms**
Controlled	Gandham et al. (2003) [15]
	Liang et al. (2010) [42]
	Farzinvash et al. (2019) [12]
Fixed/Predicted	Khan et al. (2007) [31]
	Wang et al. (2008) [89]
	Zhao et al. (2015) [94]
	Zhong et al. (2018) [95]
	Wang et al. (2019) [86]
	Wang et al. (2019) [85]
	Maurya et al. (2019) [49]
	Renold et al. (2019) [67]
	Huang et al. (2019) [21]
Uncontrolled/Random	Lin et al. (2006) [43]
	Karenos et al. (2007) [29]
	Hamida et al. (2008) [18]
	Truong et al. (2010) [80]
	Kinalis et al. (2014) [32]
	Wang et al. (2018) [87]
	Irish et al. (2019) [26]

Table 2.2: The classification methods of each individual mobile sink algorithm

MILP problem is shown to be NP-hard. Due to the high complexity, a scalable heuristic algorithm is developed for calculating the distance of the mobile sink. This algorithm consists of three stages and uses the sojourn tour method. The proposed heuristic algorithm was compared with two other heuristic algorithms (Sorted Short Tour, CSPLI algorithm) for evaluation using the lp_solver software package. The experiments adopted the parameters of real sensors, the MICA2 motes. The results show that the heuristic algorithm was nearly optimal with performance around 94% in respect of a much shorter running time.

Farzinvash et al. [12] presented a distributed and energy-efficient MS-based data gathering algorithm for collecting emergency and normal data, called Energy-efficient Emergency Data Collection (EEDMS). The algorithm combines two collection techniques based on the data type. The data collected are divided into normal and emergency data. The collection process of emergency data is done with the use of spanning tree and a static sink. However, the collection process of normal data is employed with mobile sinks. Both collection process are performed concurrently. The operating time of the network is divided into rounds, where each round consists of the configuration phase and data collection phase. To evaluate the proposed algorithm, it is compared with FTVBT, EAPC, Optimized LEACH, WRP and CB, using MATLAB. The results show that EEDMS outperforms the other algorithms in terms of lifetime and the delay of emergency data. EEDMS is able to have results reduced to 14%-59% than the other algorithms in terms of average energy consumption and around 11%-31% less than the other algorithms in terms of energy consumption for transmitting controlled packets.

Khan et al. [31] presented the CAEE (Congestion Avoidance and Energy Efficient) routing protocol for avoiding congestion and increase the lifetime of the network using a mobile sink. The mobile sink uses a fix path along the periphery of the network to collect the data from the cluster heads (CHs). In this paper, the terms "collector nodes" and "mini-sinks" are introduced. Data collector nodes are the CHs, which are created by the mobile sink and are the ones that create the clusters, which are called mini-sinks. The first trip of the mobile sink around the network is for selecting the data collector nodes and then the collection trip begins. The time the mobile sink will stay at the specific mini-sink is based on the amount of data it has to receive. The evaluation of the proposed algorithm was analyzed using OMNet++ simulation tool and is compared to a static sink-based routing strategy. The simulation results show that the protocol is effective in terms of avoiding congestion and the lifetime of the network is increased at nearly four times than the compared strategy of the static sink-based method.

Wang et al. [89] presented another cluster-based approach. A cluster-based mobile routing algorithm that consists of four main phases: *clustering phase*, *register phase*, *data dissemination phase* and *maintenance phase*. The mobile sink starts moving around the network during the second phase and when connected to a cluster head, it can receive all of its data. The movement of the mobile sink is created based on the Hidden Markov Model of the Random Waypoint. The algorithm was evaluated by being compared to the HCDD and TTDD using GloMoSim simulation tool, which was created by University of Los Angeles for wireless mobile network. The results show that, based on the overall lifetime and energy consumption, the entire network of the proposed algorithm has a better performance with a nearly 23% of increment.

Zhao et al. [94] proposed a centralized algorithm called Minimum Load Set (MLS) algorithm, a tree-based heuristic topology control algorithm with mobile sinks. This paper uses mobile sinks to maximize the lifetime of the network and its main idea is to divide the nodes in the network to anchor and non-anchor nodes. Anchor nodes are the nodes selected to deliver data to the mobile sink when it passes. The rest of the sensor nodes in the network, called non-anchor nodes, are responsible to transmit their data to an anchor node using single hop or multiple hops. MLS determines for every node its corresponding next node by creating a tree from a fictitious root node. The proposed algorithm was compared with MNL and LOCAL-OPT using the NS-2 simulator. The experimental results show that these algorithms can significantly outperform the two algorithms compared, with about 25% of the maximum relative load and 30% of the time of the network to lose connectivity.

Zhong et al. [95] presented an energy efficient multiple mobile sinks based routing algorithm for WSNs. The network is divided into clusters and in each one a cluster head (CH) is selected. This selection is performed at the beginning of each round and the parameters are the remaining energy and the distance towards the mobile sink. Only the CH can communicate with the mobile sink, while the other nodes communicate only with their CH, either directly or via multi-hop transmission. The proposed algorithm was evaluated using Matlab simulation, where the effect of the lifetime of the network was examined based on the amount of mobile sinks in the network. Simulation results show that when the number of mobile sinks in the network was increased, the performance of the network was improved in respect of the lifetime. As a fact, it is shown that when three mobile sinks were part of the network the algorithm had the best performance.

Wang et al. [86] presented a distributed routing scheme that uses data fusion and neural networks for WSNs. The routing scheme uses intelligent data gathering scheme with data fusion (IDGS-DF) and neural network for improving the performance of the network in data fusion conduction. The mobile sink moves towards a predefined path and broadcast the subdomain ID of the cluster in its range. With a greedy algorithm, the data are transmitted by the CH to the neighbor CH, which is directly communicating with the mobile sink. When the mobile sink moves to the last subdomain it will backtrack. The proposed algorithm was applied in forest fire detection, where the pretrained neural network is used fo each CH node by using the detected data to estimate the breaking out possibility of the forest fire. The evaluation of the proposed algorithm was performed using MATLAB simulation and was compared to LEACH and TBFT. Through extensive simulations the proposed algorithm shows better performance in terms of increased network lifetime and lower energy consumption.

Wang et al. [85] proposed an energy efficient routing scheme that uses clusters and sink mobility. Firstly, the network is divided into several sectors. Each sector selects a cluster head (CH) based on the member's weight and each member of a cluster chooses it based on its geographical information. The CH forms a chain with the use of a greedy algorithm to communicate with the mobile sink. The mobile sink moves in a circle with a constant angular velocity, which is used from the nodes to calculate the current position of the mobile sink. The proposed algorithm was evaluated using MATLAB simulation and was compared with CCMAR and ECDRA. The results show that the proposed algorithm outperforms the other two algorithms in respect to lower energy consumption and increased network lifetime.

Maurya et al. [49] proposed a delay aware energy efficient reliable routing (DA-EERR) technique that used a mobile sink for data transmission in heterogeneous WSNs. The main goal is to construct an energy efficient manner virtual ring in order to reduce collisions overhead and have easier accessibility to the current position of the mobile sink. The DA-EERR algorithm consists of three phases: network set up phase, data transmission phase and network maintenance phase. The use of mobile sink in the network mitigates the hot-spot problem as well as minimizing the end-to-end data delivery delay. The evaluation of the algorithm was conducted in Omnet++ network simulator and was compared with VGB, Ring Routing and LBRR. The results show that the proposed algorithm has better performance than the other algorithms in respect of different scenarios including different speed of the mobile sink and node density.

Renold et al. [67] proposed an energy efficient secure data collection algorithm in an unattended wireless sensor network using mobile sink. This algorithm consists of three phases: the identification of data collection points, the path planning by the mobile sink and the secure data collection. The data collection process between the mobile sink and convex nodes is done after authentication, which is implemented on an ElGamal scheme. The proposed algorithm was evaluated in the Cooja simulator by Contiki with different scenarios on various node counts under static and mobile sink and was compared with SEC-TMP and a mesh routing protocol that uses a static sink. The experimentation method was performed for different scenarios, where for ten simulation runs, metrics about packet delivery ratio, average end-to-end delay and energy consumption were obtained. The results show that the proposed algorithm provides better performance than the other algorithms in respect of the evaluated metrics and has a result of 100% resilience against threats.

Huang et al. [21] consider the problem of data collection with a mobile sink in a delay-tolerant wireless sensor network. The main goal is the creation of a cluster-based compressive data collection algorithm for WSNs with a mobile sink. The network consists of uniformly deployed sensor nodes and a mobile sink. The mobile sink moves periodically at the periphery of the sensor field in a fix path. The proposed algorithm presents two approaches in achieving their goal. The first implementation consists of five steps, the mobile sink broadcast, the cluster head (CH) election, cluster member (CM) - CH attachment, CH-CH association and uploading. This approach due to its randomness in generating CHs, it cannot guarantee an even distribution of CH in the network. For this reason, the second implementation was inspired by the observation that CHs are kept away from each other when they are evenly distributed. As a result, this approach introduces a competing mechanism in order for each CH to have a certain distance between the others. This algorithm was compared with MASP and MobiCluster algorithms using MATLAB. The results show that the proposed algorithm is more efficient and improves the network lifetime in comparison with the other two algorithms by about 1.2-2 times.

Lin et al. [43] used random mobility in order to create the Hierarchical Cluster-based Data Dissemination (HCDD) algorithm. The proposed algorithm divides all the nodes of the network into clusters and has three procedures, the *cluster construction*, the *Sink Location Registration* and the *Routing Path Maintenance and Data Delivery*. The proposes cluster-based method was compared with a grid-based method, where the results show that the grid-based method is outperformed by the cluster-based method and provides longer network lifetime and lower control overhead.

Karenos et al. [29] proposed another scheme that deals with congestion avoidance, called CoSMoS (COngestion avoidance for Sensors with a MObile Sink). This protocol is based on a congestion control and joint routing approach, and consists of two parts. The first part is a low cost and complexity routing scheme, which uses a mobile sink to consider effectively the dynamic reliability of the path. The second part is a regional load collection technique where the maximum sustainable load is estimated for each node within an area and a path. The proposed protocol was evaluated using the MantisOS, which is a multi-modal operating system. The results show that the proposed scheme can balance congestion and reliability in order to achieve higher delivery ratios while at the same time not hurting the throughput, in cases of high mobility than in low frequencies.

Hamida et al. [18] presented the Line-based Data Dissemination (LBDD) protocol for decoupling the operation for data dissemination using mobile sinks using the rendez-vous region concept. This protocol defines a vertical virtual line placed in the center of the network to divide it into two parts for all nodes to have easy access. Inline-nodes are the nodes within the boundaries of the wide line, while the rest of the nodes are called ordinary nodes. The task of this line is to act like a rendez-vous region for data storage and lookup. LBDD consist of two steps: *dissemination* and *collection*. The proposed algorithm was evaluated using the WSNet simulator and was compared with XYLS, RailRoad, TTDD and GHT. The realistic simulation results show that LBDD outperforms query-based as well as event-driven scenario approaches by presenting the best trade-off. In more detail, the LBDD presents lower energy consumption, increased communication cost, higher delivery ratio and lower delay than the algorithms compared.

Truong et al. [80] proposed an opportunistic routing scheme using uncontrolled mobile sink in WSNs during building emergencies. The scenario of this scheme considers the deployment of WSNs in a building during fires. The mobile sinks are small powered nodes that are attached to the equipment pack of the firefighters entering the building. Due to their position and the fact that the firefighters move, they can not be controlled. Each sensor in the network, routes its data from the route towards the Base Station (BS) or to the mobile sink if in range. The mobile sink also works as a temporary connection between clusters that disconnect from the rest of the network due to faulty nodes. When a cluster recognizes that it is disconnected, it starts a new policy by storing only important data and when the energy gets low the transmission stops. Data is only transmit to the BS when the mobile sink moves towards them and transmit the data to it. The evaluation has been performed in a small laboratory-based WSN and within the NS2 network simulator. Results show that on randomly damaged networks the scheme can increase the data delivery up to 50%.

Kinalis et al. [32] proposed a biased adaptive sink mobility scheme. This scheme uses a sink node that is responsible to move around the network in a problematically way in order to stop more times in areas that tend to produce more data and at the same time favor less visited areas for covering the network faster. The scheme was evaluated in the NS2 simulation with diverse network settings and was compared to existing blind random, non-adaptive schemes. The results show that the proposed method is able to reduce the latency especially in network with non-uniform sensor distribution, without to compromise the energy-efficiency and delivery success.

Wang et al. [87] presented an Enhanced Power Efficient Gathering in Sensor Information Systems (EPEGASIS) algorithm that alleviates the hot spots problem. Hot spot nodes are nodes around the area of the sink node that exhaust their energy quicker than the other nodes since they receive and forward all data to/from the sink. To send a package directly to the mobile sink, it has to be within the optimal communication distance, otherwise the package is transmitted on several relays that generate the topology of the network. The Matlab simulator was used to evaluate the algorithm by comparing it with the typical PEGASIS algorithm. Results show that the proposed algorithm reduces the average energy consumption at about one-third than the other algorithm. In general, EPEGASIS has a better performance than the typical PEGASIS in terms of network lifetime. However, the elimination of the hot spots problem is not completely.

Irish et al. [26] proposed an efficient data collection protocol named Dynamic Sink Mobility for Data Collection (DSMDC). This protocol uses a virtual grid structure, where each cell represents a cluster of nodes. Each cluster selects a cluster head (CH) based on the remaining energy of the currently elected cluster head. The mobile sink will collect the gathered data from the CH with the use of the standard SDMA method. Three sink migration algorithms are presented. The dynamic sink mobility for data collection uses the DEF that calculates the trail points that the mobile sink follows. The Detected Event Frequency sorted in Ascending (DEF-A), where the table of the mobile sink is sorted in an ascending order based on the detected event frequency. Finally, the Detected Event Frequency sorted in Descending (DEF-D), where the table of the mobile sink is sorted in a descending order based on the detected event frequency. The evaluation was conducted in the NS2 simulator where the three algorithms were compared. The results show that DSMDC presents lower energy consumption as well as higher throughput than DEF-A and DEF-D.

Discussion. The papers presented above propose solutions using mobile sink(s). These solutions are mainly focused in resolving the disconnection problem which is cost due to faulty nodes in the network. It is noticeable that most solutions use one mobile sink. The main idea is to divide the network into clusters and use the mobile sink to move at each cluster to collect the data. Cluster head nodes are usually the ones that define the path of the mobile sink and the only ones communicating with it. The cluster heads can be either selected based on a parameter or by the mobile sink. In Appendix B.1, the table of the strengths and limitations of each paper are presented. The work of this thesis uses mobile nodes that are not represented by mobile sinks. The mobile nodes have the same characteristics and abilities as the sensor nodes of the network, with the only difference of having the ability to change their location. Additionally, the mobile nodes are used to assist the existing nodes and are not used for moving around the network and collect data.

Group of Mobile Sinks

The algorithms described in this section use mobile sinks that move together as a group to complete a specific task. The different mobility patterns (Fig. 2.4) of a group and the leader selection method (Fig. 2.3) used by the algorithms of this section, are shown in Table 2.3.

Algorithm	**Leader Selection Type**	**Group Mobility Pattern**
Park et al. (2010) [62]	Predetermined	Random
Lee et al. (2010) [40]	Predetermined	Random
Park et al. (2010) [61]	No leader	Random
Mo et al. (2013) [52]	Predetermined	Random
Lee et al. (2018) [41]	Predetermined	All

Table 2.3: The classification methods of each group of mobile sink algorithm

Park et al. [62] proposed a novel communication scheme, called Mobile Geocasting (M-Geocasting), which is an extension of the traditional geocasting method for mobile sink groups in WSNs. A mobile sink group moves together but the sink members of the group can also move randomly within the restricted region. Each group has one leader sink (LS), a groupID and a sinkID. The restricted region of a mobile sink group is calculated based on the radius. The location of each mobile sink group is advertised from its leader. When a member sink moves out of the CGR, it will first select an agent node and will then inform a cache node about the location of its agent node. Hence, the data are forwarded to the member sink through the cache node and the agent node. The proposed algorithm was evaluated using Qualnet v4.0 simulator and was compared with TTDD and GMR. The results show that based on the data delivery ration and energy consumption M-Geocasting outperforms the other two algorithms with improved performance based on the flooding used for the data delivery method.

Lee et al. [40] proposed a Region Based Data Dissemination protocol (RBDD) using mobile sink groups to efficiently transmit data. Each mobile sink group selects a leader sink (LS), which is responsible to advertise the current location of the group. The paper guarantees data delivery to all the sink members of a group with the use of two mobility support approaches, *micro* and *macro* group mobility support. The micro group mobility support defines that in the group, each member is able to move outside or inside of the region individually, whereas the macro group mobility support defines the sink group as a whole that moves together. The proposed algorithm was evaluated using the QualNet 4.0 simulator and was compared to TTDD and GMR. The simulation results show that the data delivery to a mobile sinks group is guaranteed by the RBDD algorithm. In the comparison with the other two algorithms, RBDD performs better based on the restricted flooding method used for data delivery for all sinks that results in not needing frequent location updates by each sink, which also reduces its energy consumption in respect of the other two algorithms.

Park et al. [61] propose a novel strategy for data dissemination in mobile sink groups, called Cluster-based Sink Group Management (C-SGM). This algorithm has two aspects, a cluster-based virtual infrastructure, which is independent of the sinks and source nodes of the network, and an inter-cluster communication via geographic routing that uses recursive location search so that it reduces the structure construction and routing state maintenance overhead. The proposed algorithm was evaluated using Qualnet v4.0 simulator and was compared with an abstract Cluster-based Virtual Infrastructure (CVI) based scheme and LBDD. The simulation results show that C-SGM consumes a little more energy than LBDD, but provides a longer lifetime for the network, and performs better than the other two algorithms in respect to delivering data to the mobile sink group while moving.

Mo et al. [52] propose Virtual Line-based Dissemination Data protocol (VLDD) for a reliable and energy efficient data delivery for mobile sink groups. The part that differentiates this algorithm from similar algorithms is that it exploits a virtual line structure to store the data in the group region instead of flooding the data. The proposed algorithm was evaluated using the QualNet 4.0 simulator and was compared to the M-Geocasting algorithm. The results show that VLDD achieves better performance, where the energy consumption is decreased and the data delivery ratio is increased in respect of the other algorithms.

Lee et al. [41] propose an active data dissemination protocol that constructs a local data area and takes into consideration the moving direction as well as the pattern of the mobile sink group to exploit it. The leader sink (LS) will flood a query message to gather information about the member sinks location information, in order to calculate the region of the group. The proposed protocol was evaluated using Qualnet 4.0 simulator and was compared to M-Geocasting and SEAD. Many experiments took place considering specific parameters, such as the number of mobile sink in a group, the speed of the sink nodes and others. Based on these parameters, the simulation results show that in comparison to the other two protocols, this protocol has better performance, by reducing the consumed energy and enhancing the data delivery ratio.

Discussion. The papers mentioned above propose a solution with the use of mobile sinks group. These solutions are mainly focused in resolving the data dissemination problem. The main idea is to create a group region where the member sinks can move around independent and collect data, with different variations. It is shown that a leader in the group is needed to create this region by getting informed of the location of each member sink. This process creates an overhead especially with the mostly used method of flooding. Most of the papers use a predetermined and unchangeable leader selection method without getting into details of this selection and its criteria, and neither define it in its experimental parameters. In Appendix B.2, the table of the strengths and limitations of each paper are presented. The work of this thesis uses mobile nodes that are not represented by mobile sinks and act independently. Each mobile nodes is a different entities used to complete a different task. Additionally, the mobile nodes are used to assist the existing nodes and not for moving around the network to collect data.

2.2.2 Mobile Nodes

The algorithms described in this section use mobile nodes as extra resources in the network. Table 2.4 shows the different solution approaches of these algorithms concerning the use of mobile nodes.

Mei et al. [51] propose an algorithm with mobile nodes used to prevent the network from getting disconnection. This algorithm detects and reports node failure as well as replacing the failed nodes with the use of mobile robots. Three algorithms are presented that introduce four types of robots: the manager, the maintainer, the guardians and the guardees. A robot manager is responsible for receiving failure reports and assign a maintainer to handle the specific failure. A robot maintainer is responsible to receive and proceed the assignment of replacing a failed node. A robot can have both jobs. The three algorithms are compared and evaluated using Glomosim, a packet level simulator for ad hoc networks, where scenarios were implemented that differ on the number of maintenance robots in the network. The simulation results show that the centralized and dynamic algorithms have a lower motion overhead than the fixed algorithm, with a save of 10.8% in traveling distance, whereas both distributed algorithms have a higher messaging overhead than the centralized, due to its lack of scalability of the latter. Additionally, the dynamic algorithm has a repair time of 10.2% lower than the fixed and centralized.

Yu et al. [92] proposed the use of mobile nodes to reconnect the network. Based on

Algorithm	Solution
Mei et al. (2006) [51]	Replace failed nodes
Yu et al. (2006) [92]	Reconnect network with mobile nodes
Sheu et al. (2008) [74]	Replace nodes in network
Katsuma et al. (2009) [30]	Placing mobile nodes
Koutroullos et al. (2011) [36]	Alternative path of only mobile nodes
Boukerche et al. (2012) [8]	The direction-aware mobility level parameter is introduced
Jayakumari et al. (2015) [7]	Event observation in network
Vecchio et al. (2015) [82]	Area coverage with mobile nodes
Toor et al. (2019) [79]	Cluster-based routing protocol using multiple mobile nodes
Rao et al. (2020) [65]	Data transmission with mobile nodes
Pang et al. (2020) [59]	Data collection using multiple mobile nodes
Anuradha et al. (2020) [5]	Reconnect the network using mobile nodes after a fault
Bala Subramanian et al. (2021) [6]	Path planning mechanism for mobile anchors
Feng et al (2021) [13]	Mobile robots in an unknown hostile environment
Mazumdar et al. (2021) [50]	Hierarchical data dissemination with mobile data collector
Akram et al. (2021) [4]	k-connectivity restoration using mobile nodes
Papi et al. (2022) [60]	Hole recovery method with the use of mobile nodes
Wu (2022) [90]	Hole recovery method with the use of mobile nodes

Table 2.4: The classification methods of each mobile node algorithm

this idea, two algorithms are developed, a graph-oriented and a divide-and-conquer, that use as few as possible mobile nodes to solve the problem. Both algorithms were simulated and compared in order to compute the number of the required mobile nodes. The simulation results show that the graph-oriented algorithm is more complicated than the divide-and-conquer. However, the graph-oriented algorithm needs fewer mobile nodes than the divide-and-conquer algorithm in order to solve the problem, especially in a sparse density topology of nodes.

Sheu et al. [74] proposed the replacement of low energy nodes in the network with mobile nodes, without knowing the specific location of destination. The sink will assign to the mobile robot the replacement, which will start navigating in the network towards the destination. The mobile robot navigates from one node to another by monitoring the signal strength sent from the next hop node. Eventually, the mobile robot will arrive to its destination and will replace the low energy node. The navigation protocol was evaluated with three experiments performed in a large, free-space classroom. The experimental results show that the proposed protocol allows the mobile robot to navigate successfully around the sensor network with a multi-hop destination and no extra equipment. Additionally, the results show that the many-to-many service, due to signal interference, needs a longer navigation time to reach the destination than the single target service. As a result, the one-to-many service consumed fewer total energy than the many-to-many service.

Katsuma et al. [30] propose a method to solve the k-coverage problem with the use of mobile nodes placed into specific positions and at the same time extend the lifetime of the network. An area is k-coverage if the range of k sensor nodes cover each point of the area. This problem is NP hard, so a genetic algorithm is proposed to find a near optimal solution in practical time. The proposed algorithm was compared with the following methods: conventional methods, a proposed method, no balancing method, a static method and the Wang+Balancing method. The simulation results show that considering a network with hundreds of nodes, this method achieves a longer k-coverage lifetime at about 140%-190% compared to the other methods.

Koutroullos et al. [36] proposed a Mobile Congestion Control algorithm used in areas where congestion is happening permanently, for a long duration or repeatedly. The goal of this algorithm is not to replace existing routing or congestion control algorithms but to run alongside them. The main idea is to place a number of mobile nodes behind the sink node and use them when congestion occurs to create hard alternative disjoint paths in the network, in order to decrease the traffic of the congested area. The proposed algorithm was evaluated using the COOJA simulator of Contiki OS [58] and the algorithm was compared to a base scenario without mobile nodes. The simulation results show that congestion is indeed mitigated and more specifically packet loss is reduced and the throughput of the network is increased compared to the scenario without mobile nodes.

Boukerche et al. [8] introduced the direction-aware mobility level parameter. This parameter captures how fast and close to the sink node each mobile node is moving. Local, distributed data dissemination protocols are provided that exploit the node mobility in order to improve the performance. The parameter is divided into two categories. The high mobility is used for a low cost replacement for data dissemination, and the low mobility is used either to increase the data propagation redundancy or for long distance data transmissions. The protocols were evaluated using the NS2 simulation and were compared to relevant methods from the state of the art. The results show significant improvements of the proposed protocols in terms of latency, energy and delivery success.

Jayakumari et al. [7] proposed the Priority Based Congestion Control Dynamic Clustering (PCCDC) protocol in order to avoid congestion and decrease energy consumption. In this protocol, the use of mobile nodes that move around in the network and observe specific events based on certain conditions, provides complete connectivity and coverage. The nodes of the network are organized into dynamic clusters that change at each round. The proposed algorithm was evaluated using ns2 simulator and was compared with PASCC. The results show that PCCDC protocol outperforms PASCC based on 5.62% of enhanced lifetime. In more detail, PCCDC showed 42.02% lower packet loss, 17.91% lower control overhead, 5.07% lower end-to-end delay and 18% lower residual energy than PASCC.

Vecchio et al. [82] proposed a framework that uses mobile and static nodes to increase the sensing area coverage of monitoring WSNs. This technique is based on a distributed method of computing the trajectories of the mobile nodes in a greedy fashion. To improve the coverage rate, this technique uses controllable trajectories for the mobile nodes. A bidding mechanism is used for the static nodes in order to estimate the coverage holes of the network and assist in the navigation process of the mobile nodes. The evaluation of this algorithm was performed with the implementation of various scenarios using different numbers of mobile and static nodes. The proposed technique is also compared with a previous state-of-the-art technique. The results show that the proposed method is more stable and more effective in area coverage.

Toor et al. [79] proposed an energy aware cluster-based multi-hop energy efficient routing protocol, called Mobile Energy Aware Cluster Based Multi-hop (MEACDM), for hierarchical heterogeneous WSNs. The nodes are deployed in the network and divided into clusters, where cluster heads are selected based on a probability equation. This equation selects the cluster head based on the highest energy among the sensor nodes. After the formation of the clusters, the network is divided into sectors, where in each one a mobile sensor node is placed in order to collect the data from the cluster heads. The proposed algorithm was simulated in MATLAB and was compared to other cluster-based routing protocols. The results show that this algorithm has better performance based on the reduced energy consumption and hence the increased network lifetime.

Rao et al. [65] presented an SDN-based strategy for reliable data transmission in WSNs using mobile nodes. They consider a system that uses a three-layered architecture based on the SDN concept. The proposed system consists of three phases: the initialization phase, the priority and control information generation phase and the data forwarding phase. The proposed system is evaluated in simulations and results show that it outperforms the traditional approaches in respect of reliability by using mobility to reduce the number of hops. In more detail, the SDN-based strategy increased the packet transmission ratio that resulted in reducing the packet retransmission and at the same time reduced the end-to-end delay as well as the energy consumption of each sensor node, when compared to traditional networks.

Pang et al. [59] presented a method for data collection with using multiple mobile nodes for WSNs. The main idea is to use the mobile nodes as mobile sinks that move in the network and collect the data, which then return to the sink node. At first, a dynamic cluster algorithm is used to form clusters of the sensor nodes in the network. When the clusters are created, the cluster heads are selected based on the highest energy. The area is then divided into several sections based on the number of the mobile nodes available. Each mobile node is assigned to one area part to travel along the cluster head and collect the data. To optimize the path planning, the authors proposed a path-based equalization algorithm (PEABR) to adjust the path of the mobile nodes. The proposed algorithm was simulated using MATLAB and the results show a balanced path length for each mobile node that meet the constrains, as well as the reduction of the total path cost.

Anuradha et al. [5] proposed a diverse mechanism for restoring the connectivity of the network. The algorithm provides a fault node detection system, which is divided into two variations. The first algorithm is a faulty node detection method using periodic alert messages. The nodes in the network send a periodic alert message to the gateway with all the information of the node. Once the getaway receives such a message, the node is determined alive. However, when a message is not received at first, the getaway awaits the second round of alert messages to establish the fault of the node. At this point, the node is considered as faulty. The second algorithm is a faulty node detection method using residual energy. Each node is responsible to calculate its energy to determine if it is faulty or not. When the energy level of the node reaches the threshold, it is declared as faulty and informs the getaway about it. In both detection methods, the mobile relay node is informed about the faulty node and moves to the position to replace it. The proposed mechanism is evaluated in a hardware setup with Node MCU and Raspberry Pi3. The results show that the proposed approaches are suitable for resource constrained devices and provide good performance in restoring the network connectivity.

Bala Subramanian et al. [6] presented a mechanism with two mobile anchors that uses the Z-curve, an exact trajectory, to optimize their trajectory. The main idea is to use mobile anchor nodes to help the localization process of the nodes in the area and provide an optimized solution. The method consists of four levels. In the first level, the examination of the prescribe link with the range and the localization of the unidentified sensors takes place. In the second level, the transferring range of the mobile beacon are refreshed at the time of all sensor being covered for the localization. In the third level, the shortest path is obtained with the use of the mobile beacon transfer in the unknown. Finally, in the fourth level, three non-collinear unsecured beacons via the shortest path are derived by the path. The proposed mechanism was simulated in NS2 and the results show that it had better performance compared to the SCAN algorithm [37].

Feng et al. [13] presented an autonomous deployment framework for optimal localizable k-coverage strategies with the assist of a mobile robot. The localizable k-coverage is defined as the covered area within the range of the localization process of the mobile robot by k neighboring beacon nodes in the network. At first, a novel framework is presented that defines the optimal localizable k-coverage problem and preserves the connectivity and robustness for the mobile robot in the network. Then, the authors present a hole recovery method for the defined problem which is achieved by the mobile robot, that knows the concurrent mapping deployment and localization of the network. Additionally, a mapping-to-image transformation method is presented that is able to reveal the interaction between the WSN deployment and the network holes. In order to solve the defined problem, two optimality conditions are developed to maximize the coverage of the optimal localizable k-coverage in the unknown hostile environment with the use of the minimum number of sensors. The proposed framework is simulated in MATLAB by evaluating three different indoor environments: the snake-shaped, U-schaped and square-shaped. The results show it outperforming the trilaretation and spanning tree methods for the defined problem. The 27 simulation runs achieved an average rate of nearly 100% for 1-coverage, 91.34% for 2-coverage and 89% for 3-coverage.

Mazumdar et al. [50] proposed an adaptive hierarchical data dissemination mechanism for multiple mobile data collector used in dynamic WSNs. This mechanism consists of three elementary phases. The first one, called Prepare phase, a set of sojourn points is nominated from the base station and information are shared between neighboring nodes. The second phase, called Network configuration, runs subsequently with the first one. Here, the network topology is built, where it is divided into a set of disjoint clusters, and a routing tree is formed to disseminate each cluster data to a nominated mobile data collector sojourn point. In the last phase, called Steady, the data transmission is performed. Each cluster head aggregates its cluster data to the mobile data collector sojounr point. The mobile data collector visits all of its points and collects the data, which are then return to the base station. In parallel, a fault-tolerant mechanism is incorporated within the second phase to address the dynamic changes of the network during the third phase. The proposed mechanism was simulated using Python in the Spyder 3.1.2 development environment and was compared to RDLP [91], OECF [34] and EPEGASIS [88]. The results show an improved performance with respect to the performance metrics, such as network lifetime, energy and delay.

Akram et al. [4] presented an algorithm for solving the distributed movement assisted k-connectivity restoration problem in a heterogeneous network that consists of static and mobile nodes. In the algorithm, a node can identify the mobile nodes in the network and its 2-hop local subgraph. The algorithm consists of two main phases: the setup phase and the restoration phase. In the setup phase, each node determines its local subgraph and the moving cost of the mobile nodes. The restoration phase always starts when a node detects a faulty neighboring node. Upon fault detection, the node checks the effect of the fault on the networks's connectivity (i.e., whether the network is still k-connected or not) and starts constructing a path and moving process in the case the failure has indeed reduced the value of k. The neighboring node calls a mobile node to the position of the faulty node with a minimum moving cost. The minimum cost movement path between the mobile node and a neighbor of the failed node is constructed based on the locations of the nodes, moving costs, and obstacles. The proposed algorithm was compared with similar algorithms in simulation and real testbed environments. The results show the capability of the algorithm to restore the k-connectivity of the network with up to 35.5% lower sent Bytes and 40.9% lower movement cost.

Papi and Barati [60] presented a hole detection and recovery method (HDRM) for wireless sensor netowrks. The method is divided into two phases. In the first phase the detection is performed, where the existence of holes in the network are explored by the nodes itself. In the second phase, mobile nodes are used to cover the holes that are detected in the previous phase. In case that the number of holes is bigger than the available mobile nodes, the selection is done based on the highest priority and shortest distance to the mobile node. The proposed method was compared to similar methods based on different metrics with the NS2 software. The simulation results shows improved results in terms of the coverage and energy consumption.

Wu [90] proposed an efficient hole recovery method that uses mobile nodes to cover the holes in the network. This work does not separate the discovery and recovery process, but uses the connection line structure of a one-hop neighboring node of each node to determine the role of a hole boundary or hole interior node. The location of the mobile node is determined from the hole boundary node. The proposed method was compared to the bidding protocol and the sensing intersection-based hole recovery technique. Simulations results show that this method has better performance than the other techniques in terms of message transmission and complexity.

Discussion. The papers mentioned above focus on algorithms that use mobile nodes to help resolving the occurrence of a problem in the network. Their main focus is to assist the existing static nodes in either replacing them or creating alternative paths. It is shown that the most common tasks are the data dissemination method and the reconnection of a disconnected network. The most used method is the use of clusters, where the main idea is to send each mobile node to a number of cluster heads to collect data. In Appendix B.3, the table of the strengths and limitations of each paper are presented. The work of this thesis uses mobile nodes that are placed in the network to assist the existing nodes when a problem occurs. The mobile nodes do not replace the faulty nodes, but rather are positioned in a calculated location in order to serve as many affected nodes as possible.

2.2.3 Discussion

To the best of our knowledge, there are no other algorithms in the literature that solve the occurrence of a problem using the approach of this thesis. Based on the existing research in this area, the solutions that utilize mobile nodes follow two approaches. In the first approach, the mobile nodes either refer to mobile sink nodes that collect data from sensor nodes, or they are used for routing purposes. In the second one, mobile nodes are represented as mobile robots that move in the network, replacing and assisting nodes. The work of this thesis focuses only on mobile nodes, which take up the role of the sensor nodes within the network; they are not used as mobile sink nodes and are not considered as mobile robots.

It is also worth mentioning that this work addresses challenges at different layers and aspects (positioning, energy-awareness, physical transportation of nodes) as opposed to prior solutions that only address each layer separately from one another.

Chapter 3

The Node Placement Algorithm

In this Chapter we present the Node Placement Algorithms (NPA) that addresses the mobile node placement strategy and employs mobile nodes in the network to assist nodes when a problem occurs. The algorithm consists of two variations. The first variation is a local solution, called Dynamic Node Placement Algorithm, and the second variation is a solution that creates an alternative path of mobile nodes, called Direct Node Placement Algorithm.

When a node detects a problem in the network, it sends a *Problem Notification Message* (PNM) to the sink. This message contains all the information needed, so that the sink can act for mitigating the problem that appeared in the network. This information includes: its NodeID, its location and communication range, the number of packets received and forwarded per sample time period, and some neighbor table information. From the neighbor table, the following information are included: neighbor node's NodeID, hop number, number of packets received and availability flag.

When the sink node receives the PNM message it calculates the position for mitigating the problem. This position needs to be a "smart" placement for a mobile node in order to provide alternative paths to the sink and help in alleviating the area from its problem. After the calculation of the position, the sink node sends a *Moving Notification Message* (MNM) to the mobile node that is selected to assist in the problem. The MNM message includes information about the target location, the sender node's NodeID and its next hop NodeID.

When a mobile node receives an MNM message, it switches off its radio while moving towards the target location, and switch its radio back on at its new position. The use of the ON/OFF tactic is so that the mobile nodes are not detectable from the static nodes in the network while they travel towards their target locations. Finally, when the mobile node reaches its destination, it establishes a connection with its target nodes to be served.

The NPA algorithm with its two variations are described below and are used for determining the number and position of mobile nodes for mitigating a problem in the network, such as congestion.

3.1 Dynamic Node Placement Algorithm

The first algorithm proposed is the Dynamic Node Placement algorithm, referred to as Dynamic MobileCC. We use the term MobileCC to reflect the use of the MobileCC Framework of Koutroulos et al. [36]. This algorithm places a mobile node in a carefully computed position so as to alleviate the problematic area and assist the affected nodes. This mobile node can forward the packets either directly to the sink, if the sink is in its transmission range, or it can serve as a relay node and forward the received packets to other upstream nodes.

High level idea. Initially, the Dynamic MobileCC algorithm calculates the average number of packets per time unit that the defected node receives and cannot forward due to lack of buffer space. Then, it discovers the nodes that transmit their packets to the defected nodes and calculates the best position that the mobile node(s) should move to in order to receive data from a number of them. Ideally, the best position is the position where the minimum number of nodes can divert their traffic through the mobile node(s), whereas at the same time their total sending rate should be equally or more than the amount of the excess traffic of the defected node (see Alg. 1). In Appendix A.1, the flowchart of this algorithm is presented. Its operation is based on the following functions:

- Identification of defected and "defecting" nodes;
- Calculation of extra resources;
- Calculation of the position that the mobile node should be placed.

We now provide a detailed description of these functions below.

3.1.1 Identification of Defected and "Defecting" nodes

The identification of the node that is defected and the nodes that defect this node is the first step of the algorithm. This operation is normally performed by existing detection algorithms based on the problem to be solved. This information, along with the position of these nodes is communicated to the sink.

Algorithm 1 The Dynamic MobileCC Algorithm

```
1: function DYNAMICMOBILECCALGORITHM
2:     for all $n_j \in downNodes$ do
3:                         ▹ find the downstream nodes that have a higher receiving rate than AR
4:         if $numR > AR$ then
5:             add $n_j \in defNeighbors$
6:         end if
7:     end for

8:     if list_length($defNeighbors$) == 1 then
9:         ▹ find the intersection point from the line that starts from the sink and ends to node $n_j$ with the circle created from the range of the node $n_j$
10:        p = intersectionPofL&C($n_j$,sink)
11:    else
12:        for all $n_j \in defNeighbors$ do
13:            p = intersectionPofL&C($n_j$,sink)          ▹ calculate the distance between two points
14:            dist = distance2P(p,sink)
15:            add $dist \in DistanceTable$                ▹ create the distance list
16:        end for
17:                         ▹ find the point with the smallest distance to the sink from the Distance List
18:        p = getBestPosition($DistanceTable$)
19:    end if
20:                         ▹ determine if the point is in range of an active node
21:    flagyes = checkInRangeNode($p$)
22:    if flagyes == TRUE then
23:        maddr = selectMobileNodetoUse()
24:        send newPosition(p) message to $maddr$
25:    else
26:        set second_priority = p
27:        set flagsp = TRUE
28:    end if

29:    if flagyes == FALSE then
30:        set n = 2
31:        while $n \leq 7\&\&n \leq list\_length(downNodes)$ do
32:            if $n == 2$ then
33:                set i = 0
34:                         ▹ find a group of two nodes that will get assist from the mobile node
35:                for all $n_i \in downNodes$ do
36:                    for all $n_j \in downNodes$ do
37:                        totalAR = ar_$n_i$ + ar_$n_j$
38:                        if $totalAR \geq AR$ then   ▹ find the intersection point from the two circles created from the range of each nodes
39:                            p = intersectionPof2C($n_i$,$n_j$)
40:                            dist = distance2P(p,sink)
41:                            add $dist \in DistanceTable$
42:                            i = i+1
43:                        end if
44:                    end for
45:                end for
46:                if i > 0 then
47:                    pos = getBestPosition($DistanceTable$)
48:                    flagyes1 = checkInRangeNode($pos$)
49:                    if flagyes1 == true then
50:                        maddr = selectMobileNodetoUse()
51:                        send newPosition(pos) message to $maddr$
52:                    else
53:                        second_priority1 = pos
54:                        flagsp1 = true
55:                        n++
56:                    end if
57:                end if
```

```
58:             else
59:                                              ▷ find a group of more than two nodes that will get assist from the mobile node
60:                 for all $n_i \in downNodes$ do
61:                     for all $n_j \in downNodes$ do
62:                         p = intersectionPofL&C($n_i$,$n_j$)
63:                         for all $n_z \in downNodes$ do
64:                             totalAR = ar_$n_i$ + ar_$n_j$ + ar_$n_z$
65:                             if $totalAR \geq AR$ then
66:                                 isRange = checkIsRange($p, n_z$)
67:                                 if isRange == TRUE then
68:                                     count = count + 1
69:                                 end if
70:                             end if
71:                         end for
72:                         if count == n-2 then
73:                             dist = distance2P(p,sink)
74:                             add $dist \in DistanceTable$
75:                         end if
76:                     end for
77:                 end for
78:                 pos = getBestPosition($DistanceTable$)
79:                 flagyes1 = checkInRangeNode($pos$)
80:                 if flagyes1 == TRUE then
81:                     maddr = selectMobileNodetoUse()
82:                     send newPosition(pos) message to maddr
83:                 else
84:                     distp = distnace2P(pos,sink)
85:                     distp1 = distnace2P(second_priority,sink)
86:                     if $distp \leq distp1$ then
87:                         second_priority1 = pos
88:                         flagsp1 = true
89:                     end if
90:                 end if
91:             end if
92:         end while
93:     end if
94:     if flagsp1 == FALSE && flagsp==TRUE then
95:         maddr = selectMobileNodetoUse()
96:         send newPosition(second_priority) message to maddr
97:     else if flagsp1 == TRUE && flagsp==FALSE then
98:         maddr = selectMobileNodetoUse()
99:         send newPosition(second_priority1) message to maddr
100:    else
101:        distp = distnace2P(second_priority,sink)
102:        distp1 = distnace2P(second_priority1,sink)
103:        if $distp \leq distp1$ then
104:            maddr = selectMobileNodetoUse()
105:            send newPosition(second_priority) message to maddr
106:        else
107:            maddr = selectMobileNodetoUse()
108:            send newPosition(second_priority1) message to maddr
109:        end if
110:    end if
111: end function
```

3.1.2 Calculation of Extra Resources

In this step, the *Additional Resources* are calculated based on the average number of packets per time unit (seconds) that the defected node receives and is not able to forward. This parameter defines the excess traffic the mobile node will require to accommodate. In particular, for a *defected node i*, the *Additional Resources* rate $AR(i)$ is calculated by the equation:

$$AR(i) = \frac{Recv(i) - Tran(i)}{t - t_0} \tag{3.1}$$

where,
$Recv(i)$ is the number of packets that i has received from its neighbors,
$Tran(i)$ is the number of packets that i has transmitted,
t is the current time, and
t_0 is the time that i started transmitting packets.

Based on this equation, the mobile node that will move close to the problematic area, should be able to receive and forward the excess traffic load that cannot be forwarded by the defected node. Thus, the defected node will receive just the traffic it can accommodate and the problem will be alleviated.

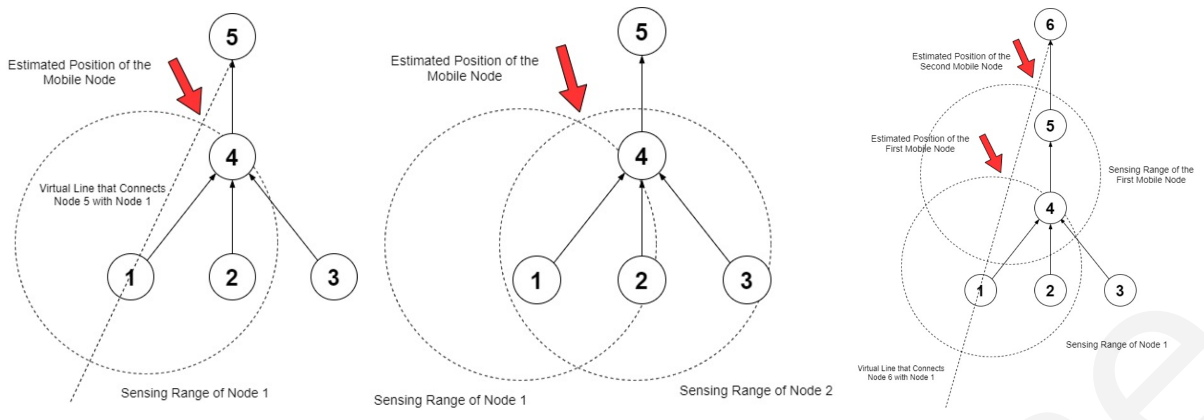
Below a more detailed description of each step is presented.

3.1.3 Calculation of the Position that the Mobile Node Should Move to

The algorithm checks whether there is a single node, which if it stops transmitting towards the defected node, the problem will be alleviated. If there is such a node then the single point where the mobile node should move is calculated (Alg. 1, lines 8-10). Otherwise, if there are more than one nodes, then for each of these nodes a specific point is calculated. The objective of this algorithm is to minimize the number of nodes that will transmit data through the mobile nodes, but their total sending rate should be equal to the amount of traffic that the defected node is not able to forward (Alg. 1, lines 11-15). Furthermore, the mobile nodes should be placed in a position where at least an upper non-defected node should exist in their transmission range to forward the data they receive, to the sink (Alg. 1, lines 17-19).

Finding the position where mobile nodes should move in order to serve one node.

The calculation of this specific point is performed as follows: Initially, the intersection points between the circle that is created by the radius of the transmitting range of the defected node


Estimated Position of the Mobile Node
Virtual Line that Connects Node 5 with Node 1
Sensing Range of Node 1
5
4
1
2
3
Estimated Position of the Mobile Node
Sensing Range of Node 1
Sensing Range of Node 2
5
4
1
2
3
Estimated Position of the Second Mobile Node
Estimated Position of the First Mobile Node
Sensing Range of the First Mobile Node
Sensing Range of Node 1
Virtual Line that Connects Node 6 with Node 1
6
5
4
1
2
3

(a) Single Node *(b) Multiple Node* *(c) Direct Path*

Figure 3.1: Node Placement Positions [56]

and the straight line that connects the sink with this node, is calculated (function *intersectionPofL&C()*). Between these two points, the point which is closer to the destination node in comparison to the point which is closer to the mobile node, is chosen. This is illustrated in Fig. 3.1a.

Let us consider as (X_k, Y_k) the coordinates of the node that is going to be served by the mobile node and (X_{sink}, Y_{sink}), the coordinates of the sink. In case that the coordinate X of this node, or Y respectively, is the same as of sink's, i.e. if $X_k = X_{sink}$, then the intersection point will be $(X_k, Y_k + node's\ Tx\ range)$ if $Y_k < Y_{sink}$ and $(X_k, Y_k - node's\ Tx\ range)$ if $Y_k < Y_{sink}$. If $Y_k = Y_{sink}$, then the intersection point will be $(X_k - node's\ Tx\ range, Y_k)$ if $X_k > X_{sink}$ and $(X_k + node's\ Tx\ range, Y_k)$ if $X_k < X_{sink}$.

For each node that has a sending rate greater than the *Additional Resources* rate, the position of the mobile node is calculated and the algorithm checks whether there is a node closer to the sink which is not defected, so as to transmit the data that it receives.

Finding the position where mobile nodes should move in order to serve more than one nodes.

If there is not any available relocation position of the mobile node suitable to serve just one node, then for a number of nodes equal to n (Alg. 1, lines 26-81), where n is a number between 2 and 6 according to [77] and [20], the following procedure is followed:

Initially, the algorithm described in [33] is employed. This algorithm identifies the subset of the nodes that transmit their data to the defected node. Only the subsets that have a total sending rate greater than the *Additional Resources* rate of the defected node are used for calculations (Alg. 1, lines 30-47 and 48-81). For each of these subsets, the algorithm finds the common point in the transmission range of the nodes, which is closer the sink. To achieve this, the algorithm considers for each pair of these nodes, the cross-section of their transmitting ranges. Then, it checks whether this cross-section point is within the transmitting range of the rest of the nodes, besides the pair under reference (Alg. 1, lines 48-65). If, for a subset of nodes, more than one appropriate point is calculated, then the point which is closer to the sink is chosen (Alg. 1, lines 66-81). This is illustrated in Fig. 3.1b.

The procedure halts when at least a common subset of nodes n is found, for $n \in [2, 6]$. If there is more than one subsets of size n, and more than one common point, then a mobile node is chosen to move to the common point that is closer to the sink. Thus, the algorithm makes sure that, from the smallest subsets ($n = 2$) to the largest subset ($n = 6$), the subset that is being served by the mobile node is the smallest. This attribute secures the validity of the first limitation of this algorithm, that the least number of nodes should change destination node.

3.2 Direct Node Placement Algorithm

The second algorithm proposed is the Direct Node Placement algorithm, referred to as Direct MobileCC. Similarly to the Dynamic MobileCC algorithm, it does not replace any existing topology control, congestion control, or routing algorithms, but runs alongside them. The difference between the two algorithms relies on the use of the mobile nodes. While the Dynamic MobileCC uses a mobile node for each problem occurrence, the solution is local, the Direct MobileCC creates a completely new and direct (disjoint) alternative path of mobile nodes towards the sink. This solution provides a faster establishment of the connection to the sink node. As our experimental evaluation shows (cf., Section 6.3.2), this helps to reduce the number of dropped packages, trading however, use of resources (and hence, energy).

High level idea. Initially, the Direct MobileCC algorithm calculates the position of the first mobile node placed in the network with the use of the Dynamic MobileCC algorithm. Then, a direct line is created that starts from the fist placed mobile node and ends at the sink. This line is filled with additional mobile nodes until a direct connection with the sink is achieved (see Alg. 2). In Appendix A.2, the flowchart of this algorithm is presented.
Its operation is based on the following functions:

- Calculation of the position of the first mobile node using the Dynamic MobileCC algorithm.

Algorithm 2 The Direct MobileCC Algorithm

```
1: function DIRECTPATHMOBILECCALGORITHM
2:                          ▷ to calculate the new position of the mobile node call the dynamic algorithm
3:     pos = DynamicMobileCCAlgorithm()
4:     flagdp = false                                                    ▷ a flag to stop the loop
5:     while flagdp == false do
6:                                          ▷ to determine if the position is in the range of the sink
7:         isSinkRange = checkinrangeofsink(pos,sink,Range)
8:         if isSinkRange == true then
9:                              ▷ to select the mobile node that is available to be sent to position pos
10:            call send_mnode(pos)
11:            flagdp = true;
12:        else
13:                              ▷ to select the mobile node that is available to be sent to position pos
14:            call send_mnode(pos)
15:                   ▷ find the interesection point from the line at pos and the circle of the sink
16:            pos = intersectionPofL&C(pos,sink,Range)
17:        end if
18:    end while
19: end function
```

- Creation of a path consisting of mobile nodes, starting from the first mobile node, that was placed from the previous function and ending at the sink.

3.2.1 Calculation of the Position of the First Mobile Node

The first mobile node is located at the position calculated by using the Dynamic MobileCC algorithm 1. If this mobile node is in the range of the sink and is able to transmit its received data directly to it, the process terminates. Otherwise, the process continues with the following step.

3.2.2 Creation of a Path Consisting of Mobile Nodes

In order to reach the sink node, it is needed to create a disjoint path. Each new mobile node placement is calculated from the algorithm so that it is in the range of the previous placed mobile node. The calculation of this specific point is performed as follows: The intersection points of the virtual circles created by the transmitting range of the initially placed mobile node and the virtual straight line between this node to the sink, is calculated. The point that is closer to the sink is the one kept. This is illustrated in Figure 3.1c. The process continues until the mobile node is in the sink.

Chapter 4

The Energy-aware Node Placement Algorithm

The NPA algorithm (see Chapter 3) stops when the mobile node is placed in the needed position and becomes active. However, at some point the current problem may be resolved and the mobile node may no longer be needed. For this reason we extended the previously mentioned algorithm (the Dynamic variation) in order to reuse the mobile nodes that are placed in the network. We call this extension Energy-aware Node Placement algorithm (eNPA), and we refer to it as Dynamic MobileCC+. To this respect, we introduce energy considerations. Based on the energy levels of the mobile nodes, they can be either reused or replaced; in the latter case, the mobile node returns to its initial position (near the sink) to recharge its battery.

When a new problem occurs in the network, there is a need to use a mobile node to resolve it. The introduction of reuse provides the opportunity to first check among the in-use mobile nodes, that is, the mobile nodes that are already placed in the network, whether one of them is available to be used. The following conditions must be applied to reuse an in-use mobile node:

1. The current problem for which the mobile node was sent to resolve, has now been resolved or the neighbor nodes of the mobile node can now find an alternative path.
2. The energy level of the mobile node does not exceed a certain threshold, which enables the mobile node to travel from its current position to the new position, and then from there to its initial position (near the sink), without running out of energy.

We now present the actions of each node type: Mobile node, Sink node and node (a static node or an in-use node acting as a relay). In Appendix A.3, the flowchart of this algorithm is presented.

4.1 Mobile Node

The mobile nodes are responsible to monitor their energy and their usage in the network. This monitoring process is done with the use of two functions: (a) the *network usage function* (see Alg. 4), which is called periodically, and (b) the *energy usage function* (see Alg. 5), which works as a push notification function.

Algorithm 3 Periodic check algorithm for mobile node m_i

```
1: flag_nu = network_usable()
2: flag_eu = energy_usable()
3: if flag_nu == FALSE AND flag_eu == OK then
4:     do nothing
5: end if
6: if flag_nu == TRUE AND flag_eu == OK then
7:     status=idle
8:     send(BecomeIdle) to sink
9:     send(Idle) to to each $n_j \in neighbor\_list$
10: end if

11: if flag_nu == FALSE AND flag_eu == TH1 then
12:     send(ShouldBeReplaced) to sink
13:     wait until Treplace OR remain_energy == TH2
14:     if remain_energy == TH2 then
15:         send(ComingBack) to sink
16:         send(GoingBack) to each $n_j \in neighbor\_list$
17:         move to init_pos
18:     end if
19:     if Treplace expired then
20:         send(GoingBack) to each $n_j \in neighbor\_list$
21:         move to init_pos
22:     end if
23: end if

24: if flag_nu == TRUE AND flag_eu == TH2 then
25:     send(ComingBack) to sink
26:     send(GoingBack) to each $n_j \in neighbor\_list$
27:     move to init_pos
28: end if

29: upon receive ("NewPosition") from sink then
30: move to new_pos
```

The procedure starts by calling the two usage functions (Alg. 3, lines 1-2). Based on their outcome, the mobile node decides its next action, as described below:

- When the node is still needed and its energy level is OK, the mobile node continues to act as a static node (Alg. 3, lines 3-5).
- When the node is not needed and its energy level is OK, the mobile node changes its status to *idle* and informs the sink node with a *BecomeIdle* message, as well as its neighbors about its new status. (Alg. 3, lines 6-10).

- When the node is needed but its energy level has reached the first threshold (warning threshold), the *replacement procedure* is triggered: the mobile node requests from the sink a replacement and waits until either the timer (*Treplaced*) expires or its energy level reaches the second threshold (departure threshold). When the timer expires, which determines a time interval in which the sink node needs to send a replacement, the node informs its neighbors about its departure and leaves (by this time, a replacing node should have arrived). However, when the second energy threshold is reached, the mobile node informs the sink node and its neighbors about its departure, and then it returns to its initial position (at the sink), to get recharged (Alg. 3, lines 11-23).

- When the node's energy level reaches the second threshold and the node is not needed (possibly it is idle), it will inform the sink node and its neighbors about its return and move back to its initial position (Alg. 3, lines 24-28).

4.1.1 Network Usage Function

This function checks whether the mobile node is still needed for the specific problem occurrence. Algorithm 4 describes this function.

Algorithm 4 Check Usable Function for mobile node m_i

```
1: function CHECK_USABLE
2:     send(FindAlternative) to each $n_j \in DN$
3:     wait until all $n_j \in DN$ to reply
4:     if $\exists reply_j == FA\text{-}SearchFailed$ then
5:         broadcast(SearchFail)
6:         return FALSE                                ▷ $m_i$ needed
7:     end if
   return TRUE                                         ▷ $m_i$ not needed
8: end function
```

The procedure starts with a (*FindAlternative*) request to each downstream node (*DN*). This request requires from a DN to search its neighbor table in order to find an alternative node to send its packets. The procedure stops, when at least one DN replies with a *FA-SearchFail message*. The mobile node broarcasts a *SearchFail message* and the function returns a *False* value, which means it is still needed. However, when all replies are *Successful messages*, the mobile node is no longer needed and the function returns a *TRUE* value.

4.1.2 Energy Usage Function

This function retrieves the current energy level of the mobile node at any time. Algorithm 5 describes this function.

Algorithm 5 Energy Usable Function for mobile node m_i

```
 1: function ENERGY_USABLE
 2:     remain_energy = listening_energy + moving_energy
 3:     return_energy = the energy needed to return to its initial position.
 4:     double_return_energy = double the amount of return_energy.

 5:     if remain_energy == double_return_energy then
 6:         return TH1                          ▷ energy reached threshold 1
 7:     else if remain_energy == return_energy then
 8:         return TH2                          ▷ energy reached threshold 2
 9:     else
10:         return OK                           ▷ energy is ok
11:     end if
12: end function
```

The function starts by calculating three parameters: the *remaining energy*, the *return_energy* and the *double_return_energy*. The *remain energy* is defined as the sum of the *listening* and *moving* energy of the node (more info can be found in Section 6.2). The *return_energy* is defined as the energy needed from the mobile node to return to its initial position (at the sink), whereas the *double_return_energy* is defined as double the amount of the return_energy (to give enough time for the replacing node to arrive). The level of the remaining energy of the mobile node defines the energy threshold returned from this function. When the energy level equals to the double_return_energy, the function returns *TH1*, which means that the first energy threshold is triggered. When the energy level drops to the return_energy, the function returns *TH2*, which means that the second energy threshold is triggered. Otherwise, the energy level of the mobile node is normal and the function returns *OK*.

4.2 Sink Node

In Dynamic MobileCC (see Section 3.1), the sink node was only responsible for assigning an idle mobile node (of those waiting near the sink) for each problem occurrence in the network. In our extended version, the sink node may choose either an idle mobile node that resides at the sink (*MList*) or one that is already placed in the network for a previous problem (*idleMList*), if there exist one. Algorithm 6 describes the tasks of the sink node.

Algorithm 6 Algorithm for sink node

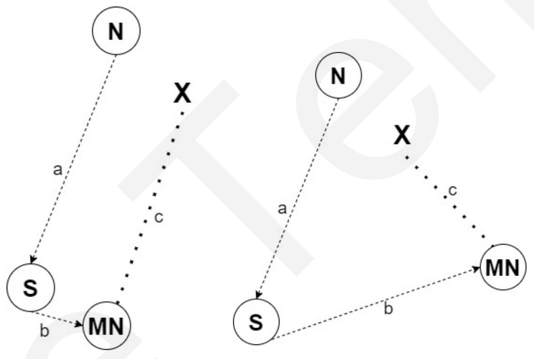
```
1:  function SEND_MNODE($new\_pos$)
2:      if $idleMList == empty$ then
3:          choose $m_k \in MList$
4:          send($NewPosition$) to $m_k$
5:      else
6:          $distnace_m$ = distance(sink_pos, new_pos)
7:          $distance_i$ = distance($closestIM\_pos$,new_pos)
8:          if $distnace_m < distance_i$ then
9:              choose $m_k \in idleMList$
10:         else
11:             $m_k = closestIM$
12:         end if
13:         send($NewPosition$) to $m_k$
14:     end if
15: end function

16: upon receive ("problem_notification") from $n_i$ then
17: new_pos = DynamicMobileCCAlgorithm()
18: call send_mnode(new_pos)

19: upon receive ("ShouldBeReplaced") from $m_i$ then
20: $new\_pos = currentpositionofm_i$
21: call send_mnode(current position of $m\_i$)

22: upon receive ("ComingBack") from $m_i$ then
23: if $m_i \in idleMList$ then
24:     remove $m_i$ from idleMList
25: end if
26: $flag_{m_i} = FALSE$
27: $current\_pos_{m_i} = init\_pos_{m_i}$

28: upon receive ("BecomeIdle") from $m_i$ then
29: $status_{m_i} = idle$
30: add $m_i \in idleMList$
```

The selection on which mobile node to send is made on the closet distance to the calculated position. Initially, the sink defines the closest idle mobile node (*closestIM*) based on its distance ($distance_i$). Then, $distance_i$ is compared to the distance of a near sink mobile node ($distance_m$). The mobile node with the smallest distance is sent to the calculated position (see Alg. 6, lines 1-15).

Another task of the sink node is to replace an in-use mobile node in case of an energy emergency. When a mobile node informs the sink about the need of replacement, the sink node assigns the position to a new idle mobile node by choosing the closest idle mobile node in the network (see Alg. 6, lines 19-21). Additionally, the sink node needs to keep track of the current status of the in-used mobile nodes. To this respect, notification messages received by the mobile nodes must be processed by the sink node. When a mobile node is about to return to its initial position, the sink node is informed in order to change its availability ($flag_{m_i}$), its position and, if necessary, remove it from the idle list (see Alg. 6, lines 22-27). When a mobile node is not needed and becomes idle at its current position, the sink node is informed and changes the status of the mobile node and it is added in the idle list (see Alg. 6, lines 28-30).

4.2.1 Analysis

We proceed to a simple analysis of the sink node algorithm, depending on whether there is reuse or not. Specifically, when a new problem occurs in the network, the *computational delay* (defined below) for the whole process is divided into two cases based on the scenario used, the no reuse scenario and the reuse scenario.



(a) Delay without Reuse *(b) Delay with Reuse*

Figure 4.1: Delay Scenarios

No Reuse Scenario (Fig. 4.1a). The notification information is sent (link a) from node ‘N’ to the sink ‘S’ and then a *moving notification* is sent (link b) from ‘S’ to a near-sink mobile node ‘MN’. ‘MN’ moves (link c) to its new position ‘X’.

Reuse Scenario(Fig. 4.1b). The notification information is sent (link a) from node ‘N’ to the sink ‘S’ and then a *reuse notification* is sent (link b) from ‘S’ to an in-used mobile node ‘MN’ in the network. ‘MN’ moves (link c) from its current position in the network to its new position ‘X’.

As a result, we can define the total computational delay or delay for short, as the sum of the communication delay (links a and b) and the moving delay (link c), i.e.,

$$delay = communication\_delay + moving\_delay.$$

The *communication_delay* is calculated as $r * dist\_hops$, where r is the *rate* of the packet per hop and *dist_hops* is the *distance* in hops, from the problematic area (N) to the sink (S), and from the sink to the mobile node (MN). For the no reuse case, in the worst case, $dist\_hops = D$, where D is the diameter of the network (and b is almost zero), whereas, for the reuse case, in the worst case, $dist\_hops = 2D$; however, the closer the in-use mobile node is to the problematic area, the smaller the moving delay will be.

The *moving_delay* is calculated as $speed*distance$, where *speed* is the speed of the mobile node and *distance* is the distance the mobile node must move from its current position to its new position. As the algorithm attempts to find the mobile node that is closer to the position to be moved (X), which could be a mobile node at the sink or an in-use mobile node, and given that in general, the moving delay is longer than the communication delay (unless an auxiliary device is used to move the mobile node, e.g., a drone), the algorithm using re-use can significantly reduce the moving delay, and hence the total delay.

4.3 In-use Node

The role of an in-use node in the network is performed by either a static node or an in-use mobile node acting as a relay node. In this extended version, the node is not only responsible for forwarding the packets it receives, but it also needs to response to requests from the mobile node in its neighborhood, which defines the *search process*.

A search process begins upon receiving a usage request from a neighboring mobile node (see Alg. 7, lines 1-2). The goal of this method is to search the neighboring table in order to find an alternative path excluding the mobile node. Two different methods are implemented (called at line 2 of Alg. 7): the *optimistic method* (see Alg. 8) and the *allocation method* (see Alg. 9), described below. Algorithm 7 shows all tasks of a static node.

4.3.1 The Search Methods

Here the two different search methods are presented. Firstly, the optimistic method is presented, where the search is only one step and then, the allocation method is presented, where a reservation of resources process takes place.

Algorithm 7 Algorithm for node n_i

```
1: upon receive ("FindAlternative") from $m_j$ then
2: call optimistic_method() OR allocation_method()

3: upon receive ("AllocationRequest") from $n_j$ then
4: Let $R$ be the data rate and $N_s$ the number of active neighbor nodes
5: $dR = \frac{R}{N_s+2}$
6: if $x \leq dR$ then
7:     $dR = dR - x$
8:     add $n_j \in ERlist$
9:     send(YES) to $n_j$
10: else
11:     send(NO) to $n_j$
12: end if

13: upon receive ("FA-SuccessMessage") from $m_j$ then
14: $flag_{n_j} = FALSE$

15: upon receive ("FA-FailMessage") from $m_j$ then
16: send(AllocationStop) to next_hop
17: next_hop = -1

18: upon receive ("AllocationStop") from $n_j$ then
19: $dR = dR + x.n_j$
20: remove ($n_j$,$x$) from ERlist
```

The optimistic method

When the node receives a *FindAlternative* request from the mobile it will search its neighbor table to find an alternative next-hop node. If the search in the neighbor table is successful, meaning that the node found at least one alternative next-hop node, it will reply to the mobile node with a (*FA-SearchSuccess*). Otherwise, the reply to the mobile node is a (*FA-SearchFail*), which means that the search was unsuccessful. Algorithm 8 describes this method. When a *Success Method Message* is received from the mobile node, the node removes the mobile node from its neighbor table and continues as normal (see Alg. 7, lines 13-14).

The allocation method

When the node receives a *FindAlternative* request from the mobile it will check its neighbor table to find from all its upper nodes at least one alternative next-hop node. Algorithm 9 describes this method. For each upper node it finds, called candidate alternative node, the node will communicate with it to ask if the candidate node can handle its extra resources.

Upon receiving a *AllocationRequest* request from a neighbor node, the candidate node will calculate its additional resources to check if this extra resources can be handled (see

Algorithm 8 Optimistic method for node n_i

```
1:  function OPTIMISTIC_METHOD($m_j$)
2:      $counter = 0$
3:      for all $n_j \in neighbor\_list$ do
4:          if $flag_{n_j} == TRUE$ then
5:              $counter = counter + 1$
6:          end if
7:      end for
8:      if $counter > 0$ then
9:          send(FA-SearchSuccess) to $m_j$
10:     else
11:         send(FA-SearchFailed) to $m_j$
12:     end if
13: end function
```

Alg. 7, lines 3-12). The request is only accepted from the candidate node only if:

$$x \leq \frac{R}{N_s + 2}, \tag{4.1}$$

where x is the amount of extra resources from the requesting node, R is the data rate of the network and N_s is the number of the active neighbor nodes of the candidate node.

If the extra resources ‘x’ are in the range of the resources the candidate node can handle, then the reply to the requesting node is an *acceptance message* and the node is added to the extra resources list (*ERlist*). However, if the extra resource cannot be handled, then the reply is a *rejection message* without any other actions. Based on the reply of the current candidate node examine, the node will either continue its search or provide a reply to the mobile node. If the reply from one candidate node is positive, the reply to the mobile node is a a *success message* (see Alg. 9, lines 7-12); otherwise the reply is a *failure message* (see Alg. 9, lines 14-16). After the search method, the node waits for the results of the procedure from the mobile node, either a *success method message* or *failed method message*. Upon receiving a *success method message*, the node removes the mobile node from its neighboring table (see Alg.7, lines 13-14) and assigns the selected node from the search as its next hop node (see Alg. 9, line 9). The latter is important, as without this assignment the normal function will indicate a random next-hop node but this will not work here as the selected candidate is the one needed. However, upon receiving a *failed method message*, the node needs to deallocate its extra resources from the selected node by broadcasting a *stop process message*

Algorithm 9 Allocation method for node n_i

```
 1: function ALLOCATION_METHOD($m_j$)
 2:     $success = FALSE$
 3:     for all $n_j \in neighbor\_list$ do
 4:         if $flag_{n_j} == TRUE$ then
 5:             send($AllocationRequest(x)$) to $n_j$
 6:             wait until $n_j$ replies
 7:             if $reply == YES$ then
 8:                 send($FA$-$SearchSuccess$) to $m_j$
 9:                 next_hop = $n_j$
10:                 $success = TRUE$
11:                 break
12:             end if
13:         end if
14:         if $success == FALSE$ then
15:             send($FA$-$SearchFailed$) to $m_j$
16:         end if
17:     end for
18: end function
```

(see Alg. 7, lines 18-20).

Analysis of the search methods

We now present a time analysis of the two search methods.

Optimistic Search method. As it can be observed from Alg. 8, this method is a one-hop task: the mobile node communicates with its neighbors and they communicate back to it. If we assume that a one-hop point-to-point communication takes one unit of time, then the number of time units T_{opt} is proportional to the number of neighbors the in-use mobile node m has:

$$T_{opt} = 2|N_m|, \tag{4.2}$$

where N_m is the set of the neighboring nodes of m. Note that if m can broadcast the message to all its neighbors within one time unit (instead of sending one message at a time to each), then $T_{opt} = |N_m| + 1$. In any case, $T_{opt} = O(|N_m|)$.

Allocation Search method. As it can be observed from Alg. 9, this method is a two-hop task: the in-use mobile node m communicates with its neighbors (N_m), and each of its neighbors i

must communicate with its own neighbors (N_i) as well. Thus, the number of time units T_{al} is bounded by:

$$T_{al} \leq 2|N_m| + 2 \sum_{i \in N_m} |N_i|. \tag{4.3}$$

The equality occurs in the worst case scenario where the nodes in N_m need to communicate with all of their neighboring nodes. Even if nodes can broadcast messages to their neighbors, it still follows that $T_{al} = O(|N_m| + \sum_{i \in N_m} |N_i|)$.

The above analysis suggests that the optimistic method is more efficient in terms of time. However, the allocation method seems to be more effective in the long run, as it ensures that the additional resources can be handled, taking into consideration what has already been "reserved" by the ongoing search mechanism (cf. Section 6.4).

Chapter 5

The Carrier-based Node Placement Algorithm

In the eNPA algorithm (see Chapter 4), a mobile node was able to move to the calculated position by itself. However, the decision of employing mobile carriers instead of mobile nodes lies on the fact that mobile nodes impose severe energy costs to the network and networks can accommodate a limited number of these nodes. As a result, it is realistic to state that each sensor node that needs to be replaced in the network is carried by either a mobile robot or a drone.

For this reason, we extended the previously mentioned algorithm (eNPA) by adding carriers that are able to pick the mobile nodes and transport them to the position that has been calculated by the sink node and contribute to the solution of a problem that has appeared in the network. We call this extension Carrier-based Node Placement algorithm (cNPA).

We now present the functionalities of each node type. In Appendix A.4, the flowchart of this algorithm is presented.

5.1 Mobile Node

A mobile node is transported to the network by its carrier for assisting the already placed nodes when a problem occurs. Once the mobile node is placed at its new position it will act as a static node, but it will also be responsible of monitoring two parameters, its energy and its usability. The monitoring of these parameters is based on two functions:

The *Energy Usage* function acts as a push notification and returns the current energy level status of the mobile node. The energy is defined as the energy needed for performing its sensing and communication activities. This function can return three possible answers based on the current energy level of the mobile node at that time. When the energy level does not reach any threshold, it returns an *OK* value. When the energy reaches the *replacement threshold*, which is defined as the energy required until the arrival of the replacement node, it returns a *TH1* value. Alternatively, when the energy reaches the *return threshold*, which is defined as the energy required for waiting the carrier to pick it up and return it to its initial place for charging, it returns a *TH2* value.

The *Usability* function is periodic and checks whether a mobile node is still required to assist the network. The procedure begins by requesting each of its downstream neighboring nodes to search for an alternative next-hop node to send their packets (*FindAlternative* request). When at least one neighbor's response is negative, the procedure stops and returns a negative value. Otherwise, when all nodes reply positively and the mobile node is not required, a positive value is returned.

Algorithm 10 Periodic check algorithm for mobile node m_i

```
1:  $flag_u$ = check_usable()
2:  $flag_e$ = energy_usage()
3:  if $flag_u$ == FALSE AND $flag_e$ == OK then
4:      do nothing
5:  end if
6:  if $flag_u$ == TRUE AND $flag_e$ == OK then
7:      status=idle
8:      send(BecomeIdle) to sink
9:      send(Idle) to to each $n_j \in neighborsList$
10: end if
11: if $flag_u$ == FALSE AND $flag_e$ == TH1 then
12:     send(ShouldBeReplaced) to sink
13:     wait until Treplace
14:     if Treplace expired then
15:         send(ComingBack) to sink
16:         send(GoingBack) to each $n_j \in neighborsList$
17:         move to initPos
18:     end if
19: end if
20: if $flag_e$ == TH2 then
21:     send(ComingBack) to sink
22:     send(GoingBack) to each $n_j \in neighborsList$
23:     move to initPos
24: end if
25: move to newPos
```

The procedure is a slight modification of the one presented in Section 4.1. It starts by calling the two usage functions (Alg. 10, lines 1-2). Based on their outcome, the mobile node decides its next action, as described below:

- Continue: the mobile node is still needed and its energy is OK, so it will continue to act as a static node (Alg. 10, lines 3-5).
- Become Idle: the mobile node changes its status to idle when it is not needed and its energy level is OK (Alg. 10, lines 6-10).

- Replace: the mobile node requests a replacement from the sink, when it is still needed and its energy has reach the first threshold. It waits for a certain time and then returns to its initial position to get charged, but firstly informs its neighbors about this action (Alg. 10, lines 11-23).
- Return: the mobile node has reached its second threshold and needs to return to its initial position for immediate charging. It informs its neighbors and sink about its need and moves back (Alg. 10, lines 24-28).

5.2 Carrier

The node type of the carrier is the novel part of this work. A carrier represents the means of transportation required to move the mobile nodes in the network. The use of a carrier in the network can be divided into two approaches: (a) stay and (b) leave, based on the action after the transportation is completed.

5.2.1 Stay Approach

In this approach the carrier moves a mobile node to its new position and waits at that position until it gets new instructions from the sink node.

Algorithm 11 Algorithm for carrier - approach stay

```
1: upon receive ("Move($m_i, pos1, pos2$)") from $sink$ then
2: if $pos1 == currerntPos$ then
3:     pick $m_i$
4:     move to $pos2$
5: else
6:     move to $pos1$
7:     pick $m_i$
8:     move to $pos2$
9: end if
```

Algorithm 11 presents the *Move* action of the carrier. Each movement is defined by the first parameters of the positions (*pos*1). The variable *pos*1 defines the first location the carrier needs to move to. In case that this position is the same as its current position (*currentPost*) of the carrier, it means that the mobile node assigned for transportation is its nearest one. Otherwise, the carrier need to move to this position and take the mobile node from there. The first case can be either a new mobile node being placed in the network or a reuse method for an in-used mobile node. The second position parameter (*pos*2) defines the second location the carrier needs to move to, which determines the destination position.

The carrier is also responsible to periodically check its current energy level in order to be able to return back and charge its battery when needed. The energy of the carrier is calculated as the sum of its listening and moving energy. When the level is OK, it continues operating as normal. However, when the energy threshold is reached, the carrier just returns to its initial position for charging, but before it informs the sink about its return, to remove the carrier from the assigned mobile node.

5.2.2 Leave Approach

In this approach, the carrier after each task is performed, always returns to its initial position (at the sink), where it waits for new instructions by the sink node while it can charge its battery, if needed. The *Leave Approach* could be used in a hostile environment, where it could be safer for the carrier to return near the sink or its initial position in the network, rather than stay at the destination position. This approach can easily modify the return position of the carrier based on the needs of the network. Algorithm 12 describes all actions performed by the carrier.

Algorithm 12 Algorithm for carrier c_i in the Leave Approach

```
1: upon receive ("Move($m_i, pos1, pos2$)") from sink then
2: if $pos1 == currentPos$ then
3:     pick $m_i$
4:     move to $pos2$
5:     place $m_i$
6: else
7:     move to $pos1$
8:     pick $m_i$
9:     move to $pos2$
10:    place $m_i$
11: end if
12:     return to initPos

13: upon receive ("Replace($m_i, pos1, pos2, m_j$)") from sink then
14: if $pos1 == nearSinkPos$ then
15:    pick $m_i$
16:    move to $pos2$
17:    place $m_i$
18: else
19:    move to $pos1$
20:    pick $m_i$
21:    move to $pos2$
22:    place $m_i$
23: end if
24: pick $m_j$
25: return to initPos
```

The following actions can be performed by the carrier:

- *Move*: this action can be divided into two tasks, *place* and *reuse*. In the place task, the variable *pos*1 is set as a near sink position, which means that the carrier will pick a new mobile node and transport it to the destination position (*pos*2), where it will leave the mobile node. In the reuse task, the carrier will have to move to the first position, where it will pick the mobile node up. It will then pick it up and move to the destination position (*pos*2). Both tasks end with the return of the carrier to its initial position near the sink.

- *Replace*: the carrier needs to transport a mobile node to a location and retrieve from there another mobile node to return it back for charging. The first part of this action is the same with the *move* task, which defines if a new mobile node or an in-use mobile node is going to be transported to the destination position (*pos*2). When the carrier reaches the destination and places the new mobile node, it will pick up the old mobile node and return to its initial position.

It is worth mentioning that in this approach, the energy level monitoring function is not needed because after each action the carrier is able to charge its battery as it returns to its initial position. Additionally, the sink node always chooses the carrier whose energy level is sufficient for the entire duration of the task, as we explain next.

5.3 Sink Node

The sink node is always responsible for selecting a mobile node and send it to its new position, either by choosing a new one or reusing an in-use mobile node. In this extension, the sink node needs to choose a carrier as well. Algorithm 13 describes the responsibilities of the sink node.
The sink node can receive the following messages that need actions:

- *Problem Notification* Message: When the sink receives such a message, it will run the *DynamicMobileCC+* algorithm (see Chapter 4) and get the position (*newPos*) for the mobile node that it will be placed. Then, it calls the *chooseMobileNodeAndCarrier* function (described in Section 5.3.1) that will return the mobile node and carrier that will be assigned to perform this task. The message about the task is sent to the carrier, including information about the mobile node and the position (Alg. 13, lines 1-4).

- *Replacement* Message from mobile node: When a mobile node needs replacement, the sink will receive a replacement message. The position for the new mobile node

Algorithm 13 Algorithm for sink s_i

```
1: upon receive ("problem_notification") from $n_i$ then
2: $newPos$ = DynamicMobileCCAlgorithm()
3: call ($m_i$,$c_i$) = chooseMobileNodeAndCarrier(newPos)
4: send (Move[$m_i, curPos_{c_i}, newPos$])" to $c_i$

5: upon receive ("Replacement") from $m_i$ then
6: newPos = current position of $m_i$
7: call ($m_j$,$c_j$) = chooseMobileNodeAndCarrier(newPos)
8: if $approach == STAY$ then
9:     send (Move[$m_j, curPos_{c_j}, newPos$])" to $c_j$
10:    if $m_i.flag_{carrier} == True$ then
11:        send (Move[$m_i, curPos_{c_i}, initPos_{m_i}$]) to $c_i$
12:    else                                                    ▷ $m_i$ has no carrier
13:        $c_k$ = chooseCarrier($newPos$)
14:        send (Move[$m_i, curPos_{c_k}, initPos_{m_i}$]) to $c_k$
15:    end if
16: else                                                       ▷ approach leave
17:    send (Replace[$m_j, curPos_{m_j}, newPos, m_i$]) to $c_j$
18: end if

19: upon receive ("comingBack") from $m_i$ then
20: if $m_i \in idleMList$ then
21:    remove $m_i$ from idleMList
22: end if
23: if $approach == STAY$ then
24:    if $m_i.flag_{carrier} == True$ then
25:        send (Move[$m_i, curPos_{c_i}, initPos_{m_i}$]) to $c_i$
26:    else                                                    ▷ $m_i$ has no carrier
27:        $c_k$ = chooseCarrier($newPos$)
28:        send (Move[$m_i, curPos_{c_k}, initPos_{m_i}$]) to $c_k$
29:    end if
30:    $status_{c_{m_i}}$ = returned
31:    $current\_pos_{c_{m_i}} = init\_pos_{c_{m_i}}$
32: else                                                       ▷ approach leave
33:    choose $c_k \in CList$
34:    send (Move[$m_i, curPos_{m_i}, initPos_{m_i}$]) to $c_k$
35: end if

36: upon receive ("BecomeIdle") from $m_i$ then
37: set $status_{m_i} = idle$
38: add $m_i \in idleMList$
```

is the same as the current position of the requesting node. Then the sink chooses a new mobile node and its carrier for the replacement and informs the carrier to bring the mobile node back (Alg. 13, lines 8). Based on the approach, different actions are needed to be performed:

- *Stay Approach*: In this approach, the sink checks if the mobile node has an assigned carrier. If this is the case, the sink informs the carrier to return the mobile node to its initial position (Alg. 13, lines 10-11). However, when no carrier is assigned, the sink will choose a carrier, either from the ones near it or the one that is closest in the network. This carrier will be assigned to return the mobile node to its initial position (Alg. 13, lines 12-15).

    – *Leave Approach*: In this approach, the carrier is responsible to take the new mobile node to the destination and from there to take the old mobile node and return it to its initial position near the sink (Alg. 13, lines 16-18).

- *ComingBack* message from mobile node: When the sink receives such a message from a mobile node, it will remove it from the idle list if it is included, and change its status into unavailable, as well as its current position to the initial position. Additionally, if the mobile node has an assigned carrier, it will change the current position to its initial and the status to “returned" (Alg. 13, lines 19-28).

- *Returning* message from carrier: When the sink receives such a message, it will change the status of the carrier to “returned" and its position to its initial position. Additionally, it will remove the carrier from its assigned mobile node (Alg. 13, lines 29-32).

- *BecomeIdle* message from mobile node: When the sink receives such a message it will change the status of the mobile node to idle and add it to the idle list (Alg. 13, lines 33-35).

5.3.1 Choose Mobile Node and Carrier Function

This function will choose the mobile node and the carrier that will assist the problem occurrence in the network. This function has two variations based on the approach used and are presented below.

Stay Approach Function (see Algorithm 14)

The stay function begins by choosing either an idle mobile node that resides at the sink (*MList*) or an idle mobile node from those that are already placed in the network for the solution of a previous problem (*idleMList*) (if such a problem exists or existed). The mobile nodes and carriers are selected based on their energy availability and the energy required to complete the specific task. The decision method is made based on the closest distance to the calculated position in the affected area. Firstly, the closest idle mobile node without a carrier (*closestIM*) and the closest idle mobile node with a carrier (*closestIMC*) are selected, based on the distance to the destination. Then, the distances of each selected mobile node are calculated, $distance_m$ for the near-sink mobile node, $distance_i$ for the idle mobile node without a carrier and $distance_{ic}$ for the idle mobile node with a carrier. The shortest distance is the parameter that will decide which mobile node is going to be sent to the new position. However, when the choice is between the two idle mobile nodes in the network, another option is reconsidered. This option will check if the carrier of the *closestIMC* mobile node can cover the distance from the *closestIM* in a shorter distance cost. When the selected mobile node does not have a carrier assigned to it, a carrier needs to be chosen. When the *CList* with the near sink carrier is not empty, a carrier from this list is chosen. However, when a near sink carrier is not available, it is required to choose a carrier from the in-use carriers. The decision is made based on the closest carrier to the starting point of the selected mobile node.

Algorithm 14 Choose Mobile Node and Carrier Function for sink s_i - Stay Approach

```
1:  function CHOOSECARRIER($pos_{m_k}$)
2:      if $Clist != empty$ then
3:          choose $c_k \in CList$
4:      else
5:          $c_k = closestC \in activeCList$ to $pos_{m_k}$
6:      end if
7:      return $c_k$
8:  end function

9:  function CHOOSEMOBILENODEANDCARRIER($newPos$)
10:     if $idleMList == empty$ then
11:         choose $m_k \in MList$
12:         choose $c_k \in CList$
13:     else
14:         $distance_m$ = distance(sinkPos, newPos)
15:         $distance_i$ = distance($closestIM\_pos$,newPos)
16:         $distance_{ic}$ = distance($closestIMC\_pos$,newPos)
17:         if $distance_m < distance_i$ && $distance_m < distance_{ic}$ then
18:             choose $m_k \in idleMList$
19:             $c_k$ = chooseCarrier($pos_{m_k}$)
20:         else
21:             if $distance_{ic} < distance_i$ then
22:                 choose $m_k = closestIMC$
23:                 choose $c_k = closestIMC_{carrier}$
24:             else
25:                 $distance_c$ = distance($closestIMC\_pos$, $closestIM\_pos$) + $distance_i$
26:                 if $distance_c < distance_i$ then
27:                     $m_k = closestIM$
28:                     $c_k = closestIMC_{carrier}$
29:                 else
30:                     $m_k = closestIM$
31:                     $c_k$ = chooseCarrier($pos_{m_k}$)
32:                 end if
33:             end if
34:         end if
35:         return $(m_k, c_k)$
36:     end if
37: end function
```

Leave Approach Function (see Algorithm 15)

The leave function begins by choosing either a near sink mobile node (*MList*) or an idle mobile node (*idleMList*) that is already placed in the network, if such exists. The decision on which node to choose is made based on the closest distance to the calculated position in the affected area. Firstly, the closest idle mobile node (*closestIM*) is selected, based on its distance from the new position. Then, its distance ($distance_i$) is compared to the distance of a near-sink mobile node ($distance_m$). The shortest distance will decide which mobile node is sent to the new position.

Algorithm 15 Choose Mobile Node and Carrier Function for sink s_i - Leave Approach

```
 1: function CHOOSEMOBILENODEANDCARRIER(newPos)
 2:     if idleMList == empty then
 3:         choose $m_k \in MList$
 4:     else
 5:         $distance_m$ = distance(sinkPos, newPos)
 6:         $distance_i$ = distance(closestIM_pos,newPos)
 7:         if $distance_m < distance_i$ then
 8:             choose $m_k \in MList$
 9:         else
10:             $m_k = closestIM$
11:         end if
12:     end if
13:     choose $c_k \in CList$
14:     return $(m_k, c_k)$
15: end function
```

5.4 In-use Node

The in-use node class which represents either a static node or an active in-use mobile node is responsible to forward the packets in the network. Another responsibility of this type of node is to reply to the *FindAlternative* request from the mobile node. When such a request is received, the node just checks its neighbor table and if it finds an available upper neighbor node, it replies with a *yes*, otherwise with a *no*. For more details, the reader is reffered to Section 4.3.

Chapter 6

The Experimental Evaluation

In this chapter, we present the experimental evaluation of the three algorithms mentioned in the previous sections. Initially, we present the evaluation setup and the evaluation metrics used. Then, for each algorithm the scenarios and their results are presented.

6.1 Evaluation Setup

To perform the experimental evaluation, we implemented our algorithms within the Contiki OS [58], an open source operating system for networked, resource-constrained systems, mainly focusing on low-power wireless Internet of Things devices. The evaluation has been performed in the COOJA simulator, a dedicated simulator for Contiki OS nodes. The specific tool was selected as it comes with the COOJA simulator that allows the creation of simulated nodes that are executing application code developed in Contiki OS and can also be uploaded to a real sensor node. The COOJA simulation is in close proximity of a real deployment. However, the COOJA mobility plugging required to define the topology positions prior to experimentation. To accommodate the feature of the algorithm and calculate the position of the mobile nodes dynamically, a script was created that was an intermediate between the algorithms and the move request to the mobile node. The algorithm sent the new position coordinates to the script through the output console of the simulation, and, in turn, sent the request of the new position.

The simulator parameters are presented in Table 6.1. Most of the parameters used are the default parameters in the COOJA simulator, which are also mapped to the parameters of real sensor nodes [58]. The total time of the simulation was selected to 15 minutes, because it is an amount that completed all task that were needed to be investigated and the network had also the chance to run after its reconnection. Since this work is related to the work from [36], the number of nodes and the initial topology of the scenarios selected was the same. The mobile node speed parameter is used in respect of the speed for the Wifibot mobile robot [96]. The maximum speed of a Wifibot mobile robot is 0.9 m/s, so the speed 0.65 m/s was selected reflecting the average speed.

Simulator/OS	COOJA/Contiki 3.0
Protocol	Contiki Multihop/Rime
MAC	ContikiMAC/CSMA
Simulation Time	15 mins
Simulation Repetition	10 times
NPA and eNPA parameters	
Emulated Mote	Tmote sky
Number of Nodes (Sink/Fixed/Mobile)	1/19/6
Mobile Node Speed	0.65 m/s
cNPA parameters	
Emulated Mote	Wismote
Number of Nodes (Sink/Fixed/Mobile)	1/19/6/6
Carrier Speed	0.65 m/s
Transmission Range (m)	25
Max Data Rate (kbps)	250
Queue Length (Pkts)	8
Packet Size (Bytes)	48
Initial Source Rate (Pkts/sec)	25
Rate Increase	50 pkts/sec every 1 minute

Table 6.1: Simulation Parameters

The network is set up and left to reach steady state for 2 minutes. All sensor nodes are equivalent to Sky Mote / Wismote nodes and bear a 10m radio range. The sources start injecting data to the network with a source data rate of 25 pkts/sec, for one minute. The data rate is then increased to 50 pkts/sec and it is constantly increased with a data rate of 50 pkts/sec, for each source, every minute. After 13 minutes each source injects to the network 600 pkts/sec, with an effective rate of 230.4Kbps. For each scenario 10 experiments were conducted each starting with a random seed.

To evaluate our algorithms, we used congestion as the case study. For this reason, we employed DAlPaS [72], a resource-control, congestion-control algorithm. DAlPaS employs a dynamic way to control topology without adding any extra load to the network. Initially, during topology control phase it builds a spanning tree from sink to source, sorting the nodes in accordance with their level (distance in hops from the sink). Thus, every node that is going to transmit data searches in its neighbor table and finds the most appropriate node, the one with the lowest level (closer to the sink) and transmits its data through this node.

When congestion occurs, each node searches in its neighbor table and finds the most appropriate node based on parameters, like level, energy, etc. The process iterates until there are no available paths from source to sink. In this case, DAlPaS algorithm stalls. This is exactly the point where our proposed algorithms start to run.

6.2 Evaluation Metrics

The following metrics were used for evaluating our algorithms: the percentage of successfully received packets, the average source to sink delay and the total energy consumption.

For each evaluation scenario, we executed 10 simulation runs. Based on the authors of [78], our algorithms are single-valued measures of performance, where the validation is performed with plots of different metrics, such as average delay, total throughput, etc. The other validation options refer to simulated and obtained data, which are obtained from the sensor nodes or other entities. This work does not consider the data captured from the sensor node, as a result a small packet, like a "hello world" packet, is send to the sink. The main factor is to reconnect the network so that the nodes are able to send packets to the sink and the sink receives them.

Percentage of Successfully Received Packets. The percentage of successfully received packets presents the successfully received packets versus all packets generated by the sources in the course of the simulation, and is calculated with the equation below:

$$Recv\_Pkts\_Ratio = \frac{successfully\_received\_packets}{total\_sent\_packets}. \tag{6.1}$$

Total Source to Sink Delay. The total source to sink delay presents the time the packet needs to be transmitted to the sink node and is calculated with the equation below:

$$Delay_{source\_to\_sink} = t_{arrival\_time} - t_{start\_time}. \tag{6.2}$$

Energy Consumption. The total energy consumption, measured in mJ, is calculated during the operation of the network, with the following equation:

$$TotalEnergy = \sum_{i=1}^{n} energy_i \tag{6.3}$$

To measure the energy consumption of the network, we calculate the energy ($energy_i$) consumed by each node i, as shown in the equation below. The general energy model used for our nodes in the network is divided into two parts. The first one is for listening and the other for moving.

$$energy_i = listening\_energy_i + moving\_energy_i, \tag{6.4}$$

where $listening\_energy_i$ is the energy computational usage and $moving\_energy_i$ is the energy usage for moving. If node i is static, then $moving\_energy_i = 0$.
Following [66], we compute $listening\_energy_i$ as:

$$\begin{aligned} listening\_energy_i = (&transmit \cdot 19.5mA + listen \cdot 21.8mA+ \\ &CPU \cdot 1.8mA + LPM \cdot 0.0545mA) \cdot 3V/4096 \cdot 8, \end{aligned} \tag{6.5}$$

where $trasmit$ is the total time of the radio transmitting, $listen$ is the total time of the radio listening, CPU is the total time of the CPU being active, and LPM is the total time of the CPU being in low power mode.
The $energy_{move}$ represents the energy usage of moving that is used for the mobile nodes (eNPA) or the carriers (cNPA). It is calculated with the following equation [96]:

$$energy_{i_{move}} = P_u \cdot \frac{s}{u}$$

where P_u is the power consumption of a given speed u and s is the total traveling distance.

Computational Time. The computational time of the algorithm refers to the time needed for the sink node to calculate the position of the mobile node. This interval can be calculated from the time the problem notification message is received by the sink until the sink node sends a moving message to a mobile node. This metric performed by the sink node, once all information is gathered, a small amount of time was required within the range of ms and 3 seconds for making calculations and provide a result. Thus, it was not included in the final results.

6.3 Evaluation of NPA

In this section, we present the evaluation scenarios and results for Algorithm NPA (Chapter 3).

6.3.1 Evaluation Scenarios

Initially, we employed 26 Tmote Sky nodes (1 sink, 19 fixed and 6 mobiles nodes) as it is presented in the topology of Figure 6.1. The Tmote Sky node category is the most simple of motes for use within a WSN and ideal for initial configurations within a COOJA simulation.

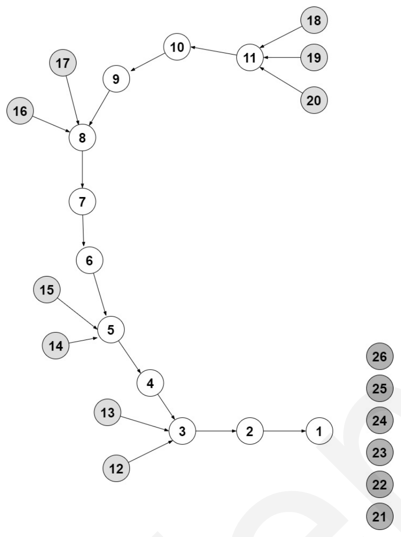
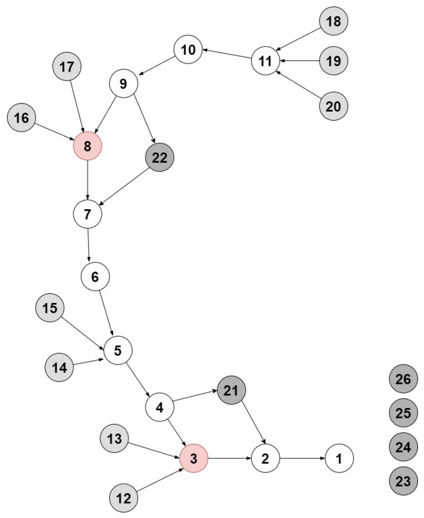


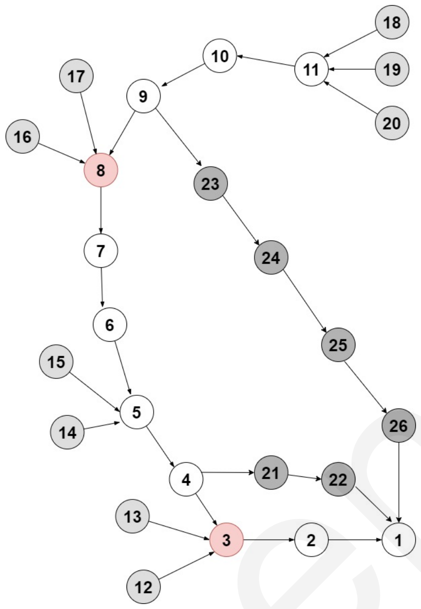
Figure 6.1: Initial Topology

In this scenario there are 9 source nodes (nodes 12-20, white), 10 relay nodes (nodes 2-11, light grey) and 6 mobile nodes (nodes 21-26, dark grey). The mobile nodes are initially placed near the sink in a sleep mode until the moment that are required by the network, as described in [35]. It is clear that based on the topology, two nodes (3 and 8) become congested due to their placement in the network. Their location is critical as these nodes are receivers from many paths and at the same time are the nodes that provide the path towards the sink.

Figure 6.2a presents the topology, after the sink calls the Dynamic MobileCC algorithm. In this scenario, two mobile nodes are employed, one for each congestion occurrence.



(a) Dynamic MobileCC Execution of the Example



(b) Direct MobileCC Execution of the Example

Figure 6.2: Execution example of the NPA variations

In Figure 6.2b we present the topology after the sink calls the Direct MobileCC algorithm. In this case two alternative mobile node paths are created. The first path consists of two mobile nodes and the other one of four mobile nodes. The different number of mobile nodes used for each path is related to the distance of the congested node from the sink. Cumulatively, this algorithm employs six mobile nodes for the creation of two disjoint paths to solve the congestion problem.

This simple experiment demonstrates that both Dynamic and Direct MobileCC algorithms can solve the problem locally. Both algorithms must employ at least one mobile node for each congestion occurrence in the network. This example indicates that Direct MobileCC needs more mobile nodes than Dynamic, which is expected since the former implements a full path of mobile nodes, from the congested point to the sink.

6.3.2 Evaluation Results

For the execution example in Section 6.3.1 we also present some basic experimental results.

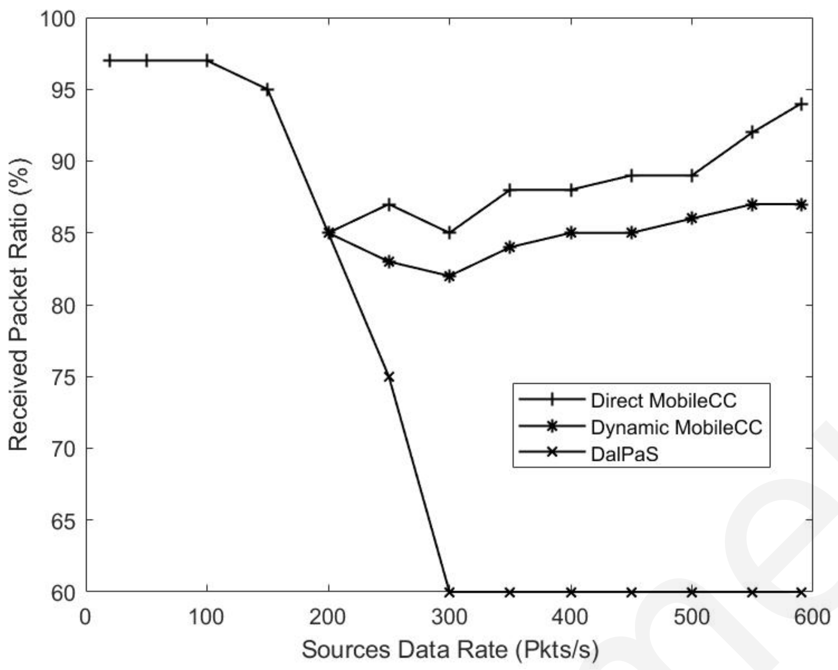


Figure 6.3: Percentage of Successfully Received Packets

In Figure 6.3 we present the percentage of successfully received packets. We observe that as the network load (i.e., sources' data rate) increases, there is a point, when the data rate is at 100 pkts/sec (i.e., $100 \cdot 48$ bytes/sec = 38.4K bits/sec) at which the DAlPaS algorithm starts failing to find alternative paths in the existing topology and the network experiences congestion. At higher network loads (above 150 pkts/sec) the network actually disconnects due to congestion (several energy depleted nodes). This is the point in the simulation, when the MobileCC algorithms are initiated.

The breaking point, when essentially no packets reach the sink, is at a source rate of 300 pkts/sec (115.2Kbps). It is interesting to note that this rate, which is roughly half of the nominal link rate matches the theoretical results on network capacity found in [71].

Our results show that, when engaged, both Direct MobileCC and Dynamic MobileCC can relieve the network from the congestion occurrence and maintain at a high level the packet transmissions. The Direct MobileCC algorithm manages to recover from congestion and recover to a received packet ratio of 94%. This is just 3% less than the original 97% achieved with no congestion.

It is worth mentioning that Direct MobileCC delivers more packets than Dynamic MobileCC. This was expected, since Direct MobileCC creates new disjoint paths of mobile nodes to the sink. In this case, any new appearance of congestion hotspots through this path is avoided. On the other hand, the Dynamic MobileCC algorithm places just the required number of nodes in specific points of the network, targeting the creation of new paths and routing traffic through nodes that were not initially accessible. In such a case, congestion may re-appear, especially in cases where some of these nodes are already in use by other flows.

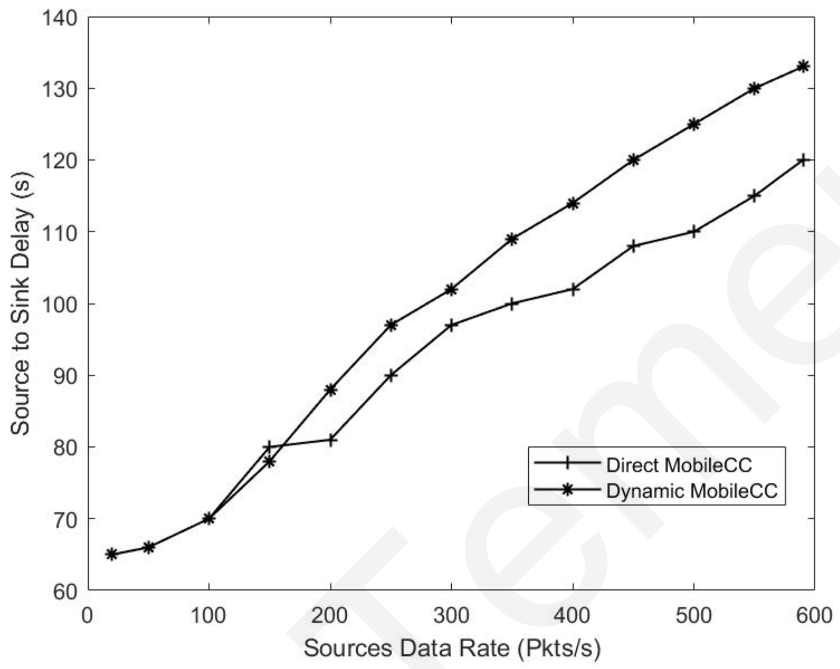


Figure 6.4: Source to Sink Delay

In Figure 6.4 we present the total source-to-sink delay in the network. In this plot, we notice that both algorithms have a total source-to-sink delay that increases as a function of the source data rate. This is expected due to the fact that collisions exist in the network, and until the network stabilizes with the help of the mobile nodes many packets are either being re-sent or sometimes are even dropped. As mentioned before, Dynamic MobileCC places only mobile nodes in positions where paths are created from existing nodes in the network so the delay is higher in comparison to Direct MobileCC that creates a new path with mobile nodes.

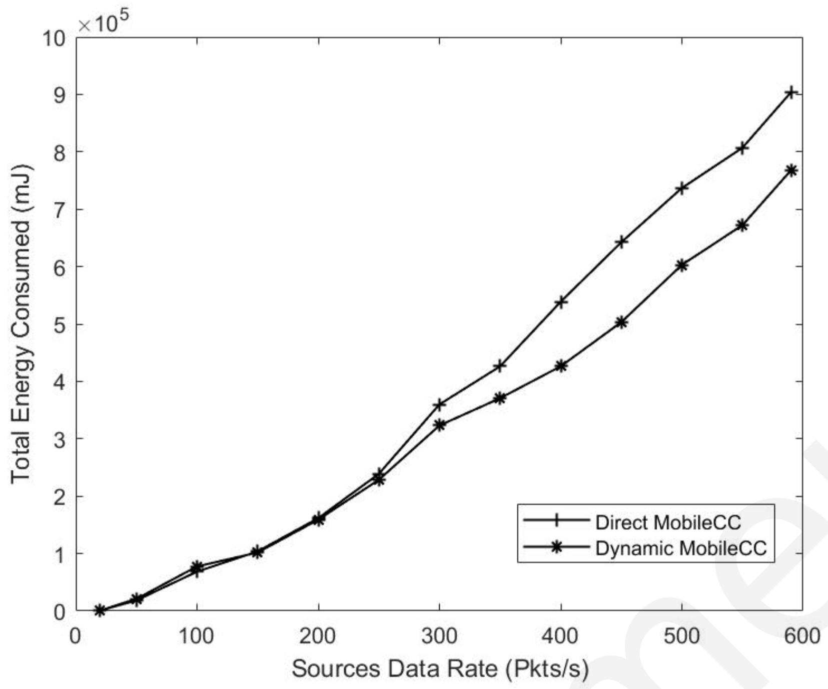


Figure 6.5: Total Energy Consumed

In Figure 6.5 we present the total energy consumed, measured in mJ, during the operation of the network. In this plot we observe that both Direct MobileCC and Dynamic MobileCC have a stable increment based on the total packets injected in the network. In comparison, Direct MobileCC has higher energy consumption than Dynamic MobileCC. That was expected, as Direct MobileCC injects more mobile nodes in the network by creating new alternative paths consisting of only mobile nodes.

Discussion

In general, based on the results it is observed that the Dynamic MobileCC algorithm uses a local solution and employs less mobile nodes in the network. On the other hand, the Direct MobileCC algorithm needs more mobile nodes to create the paths, which consists only of mobile nodes, but has better results because of the path creation that does not need time to establish new communications in the area, it just directly forwards packets to the sink node. As a result, the Dynamic MobileCC algorithm trades performance for resources. So, if limited resources are available, Dynamic MobileCC is to be chosen, otherwise Direct MobileCC would be a better choice.

6.4 Evaluation of eNPA

In this section, we present the evaluation scenarios and results for Algorithm eNPA (Chapter 4).

6.4.1 Evaluation Scenarios

Initially, we employed 26 Tmote Sky nodes (1 sink, 19 fixed and 6 mobiles nodes) as it is presented in the topology of Figure 6.6.

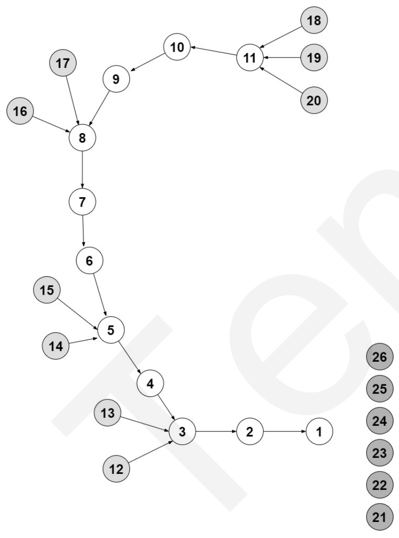


Figure 6.6: Initial Topology

In this scenario there are 9 source nodes (nodes 12-20, white), 10 relay nodes (nodes 2-11, light grey) and 6 mobile nodes (nodes 21-26, dark grey). The mobile nodes are initially placed near the sink in a sleep mode until the moment that are required by the network, as described in [35]. It is clear that based on the topology, two nodes (3 and 8) become congested due to their placement in the network. Their location is critical as these nodes are receivers from many paths and at the same time are the nodes that provide the path towards the sink.

In this subsection, we present the three scenarios used for the evaluation process of eNPA, i.e., Dynamic MobileCC+. The scenarios are divided as follow:

- Scenario 1: it presents a no-reuse scenario, which essentially evaluates the Dynamic MobileCC algorithm. In this scenario it is assumed that the mobile nodes placed in the network are not able to be reused for a new problem. This scenario is used for comparing the previous algorithm with the new version.

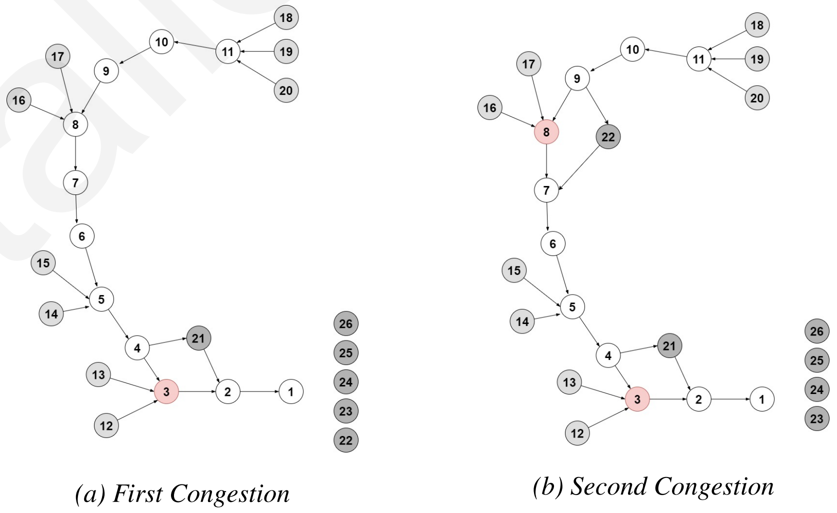
- Scenario 2: it presents a reuse scenario. In this scenario it is assumed that each mobile node that is placed in the network can be reused if all conditions are met. This scenario is used to present the benefit of reuse, which is the new addition of the algorithm.

- Scenario 3: it presents an energy exhaustion scenario. It is divided into two cases: (A) return scenario, where the mobile node is not needed in the network, but it needs to return to charge its battery, and (B) replace scenario, where the mobile node is still needed in the network and its energy level reaches its lowest threshold, and needs to be replaced.

All scenarios presented use the two search methods. As a result, each scenario has to cases: (1) Optimistic Method, where the optimistic search method is used, and (2) Allocation Method, where the allocation search method is used.

Scenario 1

In this scenario, no mobile node is reused as the problem that occurred in the network is permanent. Specifically, the congested node runs out of battery due to heavy congestion. The mobile node sent to solve this problem will need to stay there until the end of the simulation. As a result, if any other congestion occurs in the network the sink node will need to send another mobile node. This scenario represents the Dynamic MobileCC (Alg. 1), where the mobile node is just placed in the network and works as a static node and no reuse is possible.



(a) First Congestion

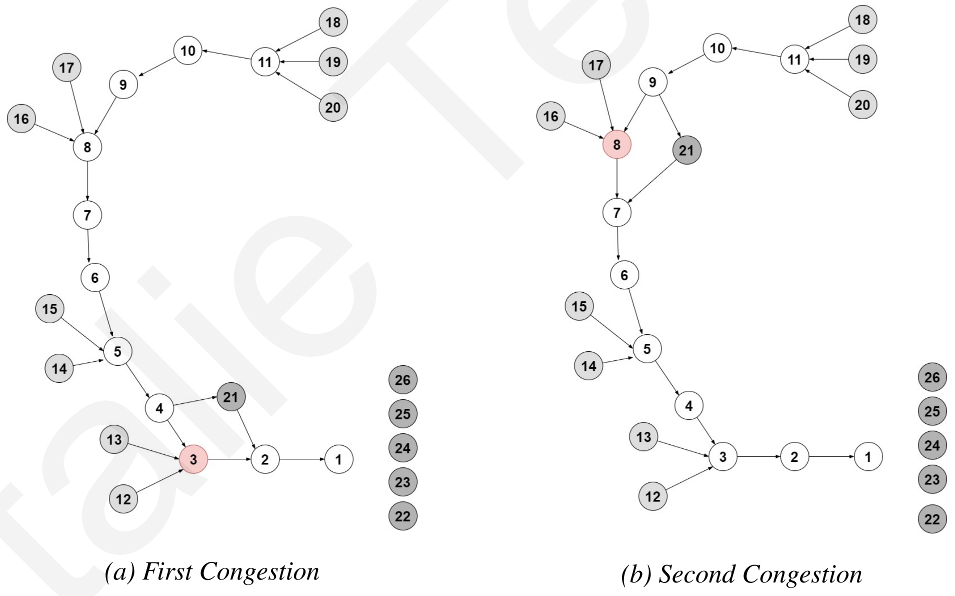
(b) Second Congestion

Figure 6.7: Scenario with no reuse

In Figure 6.7, we present the topology of this scenario. Here, two mobile nodes are employed, as a result of the permanent failure of congestion node 3. Mobile node 21 is required in the network and at each usability check, meaning that the mobile node checks if it is still needed, it gets a negative answer, which results in being active during the whole experiment. When congestion occurs at node 8, the sink node can only choose from "near sink nodes" and for this reason it sends mobile node 22 to help with the new congestion occurrence.

Scenario 2

In this scenario, a mobile node is reused. The congestion problem for which the mobile node was sent to solve at some point is resolved and as a result the congested node becomes again active. At a usability check of the mobile node, the answer of its downstream nodes is positive and the mobile node becomes idle at its current position. When a new congestion occurs in the network, the sink node needs to choose from its near-sink nodes and the idle mobile node in the network. Since the idle mobile nodes are closer to the congestion area, the sink node selects this node to move to the new position.



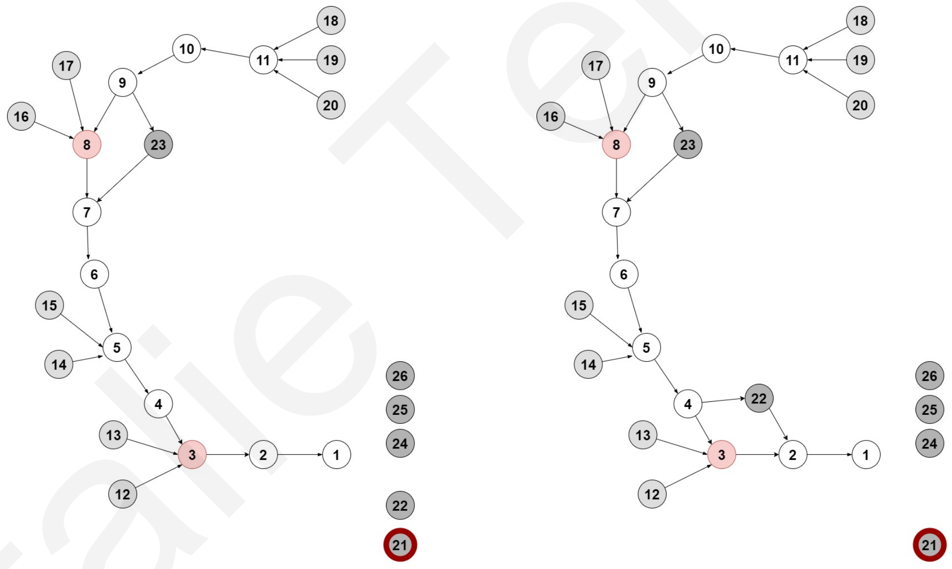
(a) First Congestion *(b) Second Congestion*

Figure 6.8: Scenario with reuse

In Figure 6.8, we present the two topologies of this scenario, the "before" and "after" the reuse of the mobile node. In this scenario, the congested node 3 will at some point resolve its congestion problem and become active again. When the mobile node 21 checks its usability, its downstream node has an available node and accepts its change. Therefore, mobile node 21 is free to become idle at its current position, by informing the sink node and its neighbor nodes. When node 8 becomes congested, the sink node has to choose between the near-sink nodes and the idle node in the network. As now the mobile node 21 is closer to the congested area, the sink node chooses the idle node and mobile node 21 moves to its new position.

Scenario 3

In this scenario the mobile node in the network is required to return to its initial position due to the lack of energy. This scenario can be implemented in two cases. The first case (*Scenario 3A*) is where the mobile node is not needed and becomes idle at some point. When its energy reaches its lowest threshold it will just send an information message to the sink, to change the mobile node's status, and return to its initial position. The second case (*Scenario 3B*) is where the mobile node is still needed in the network but its battery reaches the lowest threshold. The mobile node informs the sink node that it needs replacement and after a certain time it moves back to its initial position. In both cases, the mobile node that returns is not reconsidered for being used again, until its battery is fully charged.



(a) Returning Scenario(3A) *(b) Replacement Scenario (3B)*

Figure 6.9: Scenario of returning back

Figure 6.9, presents the topology of the cases of this scenario. In Figure 6.9a, the mobile node moves to its current position without causing any problems, whereas in Figure 6.9b the mobile node needs a replacement to move back. When the sink node sends mobile node 22 to replace mobile node 21, mobile node 21 moves back to its initial position to charge its battery. In both scenarios, the new congestion problem is resolved with a new mobile node, mobile node 23, to be injected in the network.

These experiments demonstrate the reuse of mobile nodes in the network. Reusing the mobile nodes provides the potential of using less energy and results in an energy-efficient solution for the entire network.

6.4.2 Evaluation Results

For the scenarios presented in Section 6.4.1, we also present some basic results. Each scenario described is executed for both search methods, the optimistic method (see Section 4.3.1,Algorithm 8) referred as *OM*, and the allocation method (see Section 4.3.1, Algorithm 9) referred as *ALM*.

The plots of percentage of successfully received packets and delay start at 600pkts/s load in the network. This happens because this is the point where the different scenarios execute their algorithm and difference can be shown. From the beginning of the simulation until this point the results given are the same, which is expected, as the algorithm of all scenarios until then, run the same commands.

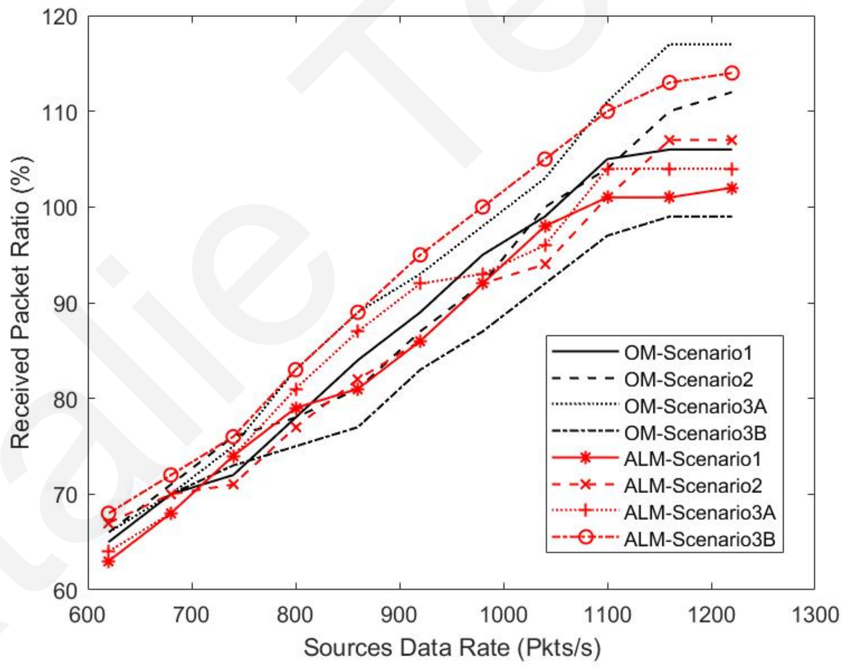


Figure 6.10: Percentage of Successfully Received Packets

In Figure 6.10, we observe that based on the methods used to find an alternative node when requested from the mobile node, the results of the *Allocation Method* are slightly better than the Optimal Method. This is normal due to the nature of the two methods, since the allocation method is more strategic and makes use of all information in the neighbor table. By accepting an allocation request it guarantees that the mobile node will not be further required. On the other hand, the optimal method does not require the node to search for an alternative node by examining exhaustive their neighboring node, in order to make its decision. This decision does not always comply with the whole network information. In both methods, it is shown that the worst scenario is the one where the mobile node needs a replacement. This is normal and accepted, because the replacement of the node will incur a delay in the routing process until the process is accomplished. The best scenario is the one where the node becomes idle and at some point will need charging and then will return to its initial position. The scenario where the mobile node is reused has definitely better results than the one where no reuse is done. This is expected, because the mobile node needs more time to move from the near sink position in comparison with the node that is already in the network.

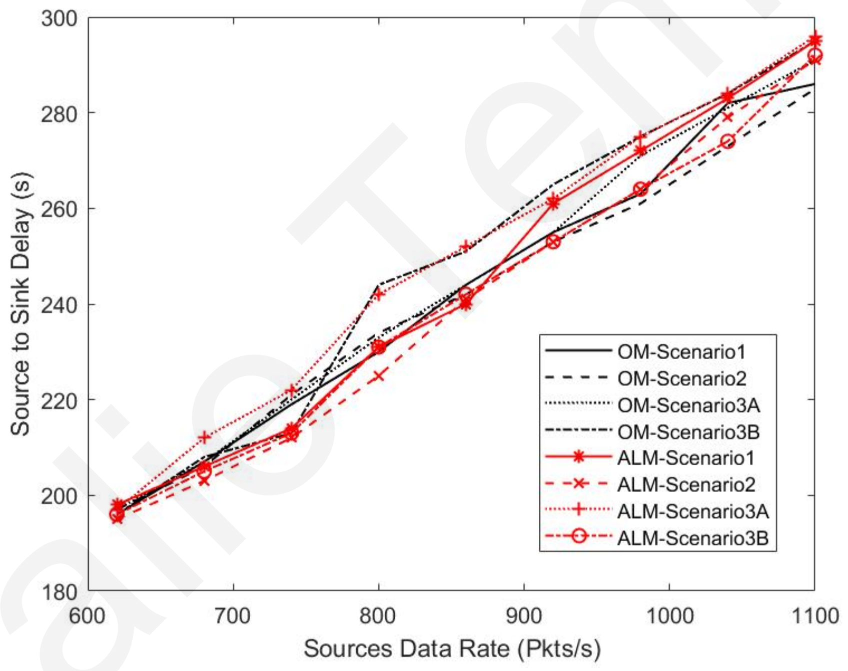


Figure 6.11: Source to Sink Delay

Figure 6.11 shows the total source to sink delay in the network. It is noticeable that both methods have an increased delay during the simulation. This is expected due to the fact of the collisions that exist in the network where many packets are retransmitted or sometimes dropped. The scenarios with the replacement of the mobile node has the most delay, which is expected due to the fact that more information is injected in the network to accomplish the replacement. The scenarios in which mobile nodes are reused have the lowest results, which is normal due to the fact that the relocation of the mobile node saves time and fewer packets are re-sent or dropped.

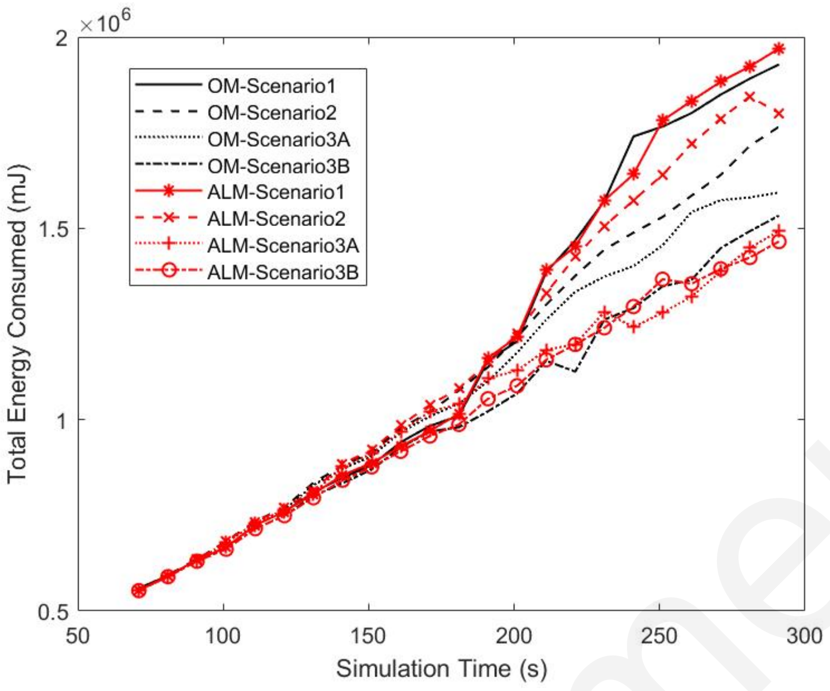


Figure 6.12: Total Energy Consumed

In Figure 6.12 we observe that both methods act similarly in response of energy consumption. However, it is noticeable that at all scenarios, the allocation method slightly consumes more energy than the optimal method, which is acceptable due to the nature of the algorithm sending more information messages in the network. It is shown that after 150 seconds is the time where each scenario has an individual execution with different outcome. The scenario with the most consumed energy is the one where no reuse is employed. This can be explained due to the fact that the mobile node inserted in the network will operate until the end and new congestion issues will be handled by another mobile node. The scenario with the least consumed energy is the one where the mobile node is not any more required and returns after its battery is exhausted to its initial/charging location. This scenario at some point operates with one less node in the network, which justifies the results. In the scenario of reusing the mobile node already placed in the network, the results are acceptable, where it result is between the no reuse scenario and the one with the returned mobile node. Comparing it with its opposite scenario, the energy consumed in this scenario is less, because mobile nodes are reused in the network in respect with the scenario of no reuse that uses more mobile nodes.

Discussion

In general, based on the results it is observed that by reusing an idle mobile node in the network will resolve the new problem faster and use less resources, in respect of a scenario that does not reuse mobile nodes that are already placed in the network, such as Dynamic MobileCC algorithm. Additionally, being able to replace an exhausted mobile node is an advantage that comes with a cost over an overhead that is the time needed to complete the replacement task. However, once the new assignment is established the problem is resolved as a new problem with the best solution coming either from a near sink mobile node or an idle mobile node in the network. Finally, in comparison between the two search method used, it is noticeable that even though the Allocation Method guarantees better results it need more time to complete the task, whereas the Optimistic method is faster but may cause a new problem in the future of the simulation, as its calculations are not verified by the candidate nodes.

6.4.3 Evaluation of eNPA with Different Energy Models

In this section, we evaluate the eNPA algorithm with different energy models. Initially, we present the energy models used for the evaluation. These models are based on different types of mobile robots. Then, we present the evaluation results, which are divided into evaluation based on each energy model and based on each scenario.

Energy Models

In this work, we focus on the energy consumption of nodes used for moving. The mobile nodes need to use their energy as efficient as possible in order not to waste too much energy while moving, and hence preserve more time for their operational time. We will present three different moving energy models from the literature, each being used by a different mobile robot.

Moving Energy Model 1. In [96], Zorbas et al. model the power consumption of a specific mobile robot. This work presents a model created by experimental results based on different speed and acceleration levels.

The mobile robot used for this work is a Wifibot [63]. This mobile robot consists of a four-wheel drive chassis controllable, infrared sensor, a web camera, a WiFi adapter, a core and an embedded system that can be one of the following systems: Linux Ubuntu, NVIDIA or Raspberry PI, as well as a free WiFi access point. The embedded system of the robot refers to the motherboard where all peripherals are connected to. In order to reduce power consumption, a low consumption power unit and a flash disk is used. A serial port is used for the communication between the embedded computer and the motor board. The role of the motor board is to play the microcontroller and the power regulator. The motor board connects to the power supply, where the power is distributed to the microcontroller and the other components of the robot.

The experiment setup included a mobile robot (Wifibot), a power analyzer that was connected to the robot, a monitor and a keyboard. The keyboard was used for the commands and the monitor for displaying the outputs. All experiments took place on a flat surface of a clean non-slippery parquet-style floor, where the robot was protected of spinning or slipping. The models built use different speed and acceleration levels and the experimental results show the relations between the energy and the speed, as well as the distance.

A mobile robot's power consumption is the sum of the power consumed by the motors and the embedded devices. The former represents the mechanical power which is used for accelerating and maintaining a constant speed.
The total energy equation is given below.

$$P_{total} = P_e + P_l + P_m,$$

where, P_e represents the power consumption of the embedded devices, P_l represents the power loss of the transformation from electrical to mechanical energy, and P_m represents the mechanical power and is given by: $P_m = mau + g\mu u$, where u is the robot's speed, m is its mass, μ is the ground friction constant and g is the acceleration of gravity.

The idea of moving is that the mobile robot starts from its initial position and accelerates until it reaches its maximum speed, where it continues with a constant speed.

This work divided the recorded power of the mobile robot into two parts: (a) the acceleration power and (b) the power while maintaining a constant speed. Based on this, the total moving energy consumption is calculated with the following equation.

$$E_u = P_{acc_u} t_{acc_u} + P_u \frac{s - s_{acc_u}}{u},$$

where, P_{acc_u} is the power of acceleration at a given speed u, s is the total traveling distance, s_{acc_u} is the distance traveled during acceleration, t_{acc_u} is the acceleration time and u the speed of the robot.

The results show that the energy cost increasing up to 66% when the robot stops frequently due to accelerations. Additionally, at higher speeds, the robot achieved high energy efficient. Although all results are based on a specific mobile robot, the main conclusion is that acceleration is an action that consumes a high amount of the energy, which results in decreasing the operation time of the mobile robot.

Moving Energy Model 2. In [19], Hou et al. present a novel energy model for mobile robots that can be used to calculate and predict their energy consumption. The main idea is to provide an energy model to be used for energy efficient strategies.

The mobile robot used for the experimentation of this work is a Mecanum. A Mecanum [10] is a four-wheeled omni-directional mobile robot. An omni-directional characteristic describes the ability of moving instantaneously at any direction without considering the configuration. This type of robot is able to move sideways, follow complex trajectories and turn on the spot, as well as perform tasks with both static and dynamic obstacles. This mobile robot uses mecanum wheels, which are similar to the universal wheels except of their rollers being mounted on angles.

The energy consumption of the robot is divided into three parts: (a) the sensor system, (b) the control system and (c) the motion system. Each part defines its own energy consumption and all together define the total energy consumption of the mobile robot.
The energy consumed by the sensor part is almost stable and is defined by multiplying the electrical power (P_{sensor}) and time (Δt). The equation is given below.

$$E_{sensor} = P_{sensor} \cdot \Delta t$$

The energy consumption of the control system depends on the power of the control circuit board and is related to the robot's running state. A robot's running state can be divided into three states: the standby state, the start to move state and the smoothly run state. Each one state has its own energy consumption formula, which is given below, and is used based on the current state of the robot.

$$E_{control} \begin{cases} E_{standby} = P_{standby} \cdot \Delta t \\ E_{startup} = \int (\phi \cdot \Delta u + (\frac{t^2}{10}) + P_{standby}) dt \\ E_{stable} = \int (P_{standby} + t^2) dt. \end{cases}$$

The energy consumption of the motion system can be divided into four parts: the traction energy consumption, the kinetic energy, the friction energy dissipation and the energy dissipated in thermal form. The motion of a robot is divided into three stages: standby, startup and stable. In the standby stage the power is constant. In the startup stage, an instantaneous pulse is needed in order to send the signal to the electric motor. Additionally, when a robot is on the move, it enters the stable operation stage. The energy consumption of the motion system is given below.

$$E_{motion} = \int P_{motion} dt = Ek + Ef + Ee + Em.$$

Each energy used in the motion energy formula given above is further explained below.

- Ek is the kinetic energy of the robot. To calculate the kinetic energy, the mass, M, of the robot and its speed, u, at the current moment are needed and is given by: $Ek = M \cdot u^2/2$.
- Ef is the friction dissipation during the robot's movement. To calculate this energy, it is needed to know the mass, the speed at the current moment and the friction coefficient, μ, between the wheel and the ground, and is given by: $Ef = \int (\mu \cdot M \cdot u) dt$.
- Ee is the energy dissipation as heat in the armatures of motors. To calculate this energy it is needed to know the time-heat constants and the speed-heat constant of the robot, as well as the time and speed, and is given by:
  $Ee = \int (\epsilon \cdot t^2 + \sigma \cdot u + \lambda \cdot t) dt$.
- Em is the mechanical dissipation that is caused by overcoming the friction torque in the actuators. To calculate this energy it is needed to know the drag coefficient of the robot itself and the vibration velocity coefficient., and is given by:
  $Em = \int \left[M \cdot e^{\zeta t} \cdot \cos\left(\psi \cdot t + u + \left(\frac{M}{2}\right)\right) + M \right] dt$.

The results show that the proposed energy model can be used by mobile robots to predict its energy consumption of its movement processes. In general, this work presents a complete model that connects all parts of a mobile robot and provides a feasible and effective model. During the experimentation results, the authors noticed that the stand by state of the mobile robot, provides in all energy consumption parts, very low numbers, which lead into re-calculating the total energy consumption considering only the sum of the idle and moving energy of the mobile robot.

Moving Energy Model 3. Many research works on energy models focus on differential drive mobile robots. This type of robots uses a drive mechanism called differential drive. This mechanism consists of two independently actuated drive wheels that are mount on a common axis. However, each wheel is able to be driven independently either forward or backwards. In order to perform a rolling motion, the robot needs to vary the velocity of each wheel but at the same time rotate a point on the common wheel axis.

We present two works that focus on different mobile robots, namely the P3-DX robot [84] and the Nomad Super Scout robot [27], respectively; these are popular mobile robots in the research community of energy consumption models. P3-DX is a mobile robot with two wheels driven by two DC motors and powered by a rechargeable battery. Nomad Super Scout robot is a two-wheel differential robot that has an embedded robot controller to control the motion commands and lower level motors.

In [84], Wahab et al. start by investigating various energy loss components of the differential drive robots and then present an energy model based on their findings. The experiments were done with a robot that has four wheels, two that are driven from the DC motors and two that act as a caster. The energy model is validated by moving the mobile robot with a specific velocity profile, where all losses have been measured and analyzed.

In [27], Morales et al. propose a power model for a two-wheel differential drive mobile robot. The model presented considers the dynamic parameters of the robot as well as the motors, and it is able to predict the consumption of the robot's energy for trajectories using variable accelerations and payloads. The experimentation was done with the use of a Nomad Super Scout II mobile robot for straight and curved trajectories. The results show that the accuracies of the energy consumption for straight trajectories are 96.67% and for curved trajectories are 81.25%.

Based on all of the above work, the following results have been obtained.

The overall energy model is given by the equation bellow after the analysis of all loss components.

$$E_{battery} = E_{dc} + E_{kinetic} + E_{friction} + E_{elect}.$$

Each energy used in the overall energy model formula given above is further explained below.

E_{dc} represents the energy produced by the DC motors of the robot. The DC motors are attached to the robot's wheel and are responsible to convert the electrical energy to mechanical energy. The conversation depends on the losses that occur, such as armature resistance loss, windage loss and stray loss. As a result, the energy consumption of the DC motors is given by the sum of the armature energy and the energy of other losses that occur. The armature loss energy ($E_{armature}$) represents the consumed energy of the armature currents and resistances of the left and right DC motors of the robot. The energy of other losses (E_{other}) represents the energy of all other losses, like friction, windage, stray etc. It is worth mentioning that the energy of other losses can be disregarded as shown by the experimental work of the authors.

The equation is given below.

$$E_{dc} = E_{armature} + E_{other}$$

$E_{kinetic}$ represents the energy loss where the output power is used in order to increase the kinetic energy and the acceleration of the robot. However, during the deceleration phase, the kinetic energy will be transformed back but due to heating a part of it will be lost. As a result, the kinetic energy consumption uses the linear ($u(t)$) and angular ($w(t)$) velocities of the robot, its mass (m) and the robot's moment of inertia (I). The equation is given below.

$$E_{kinetic} = \frac{1}{2}(mu(t)^2 + Iw(t)^2),$$

where, u is the linear velocity of the robot and is given by $u = \frac{r(w^R+w^L)}{2}$, w is the rotational velocity of the robot and is given by $w = \frac{r(w^R+w^L)}{2b}$, where r is the ratio of the robot's wheel and b is the axle length.

$E_{friction}$ represents the losses due to friction. The wheels of the robot face friction due to the cause of slight deformation of the ground or the wheel at the point of contact and can be primarily the rolling friction or rolling resistance. The equation is given below.

$$E_{friction} = \int (P^R_{friction} + P^L_{friction})dt,$$

where, $P_{friction}$ is the total power lost against friction and is given by $P_{friction} = P^R_{friction} + P^L_{friction}$, such that $P^R_{friction}$ and $P^L_{friction}$ are the power lost against friction for the right and left motor of the robot and are given by $P^R_{friction} = \mu mg(u(t) + bw(t))$ and $P^L_{friction} = \mu mg(u(t) - bw(t))$.

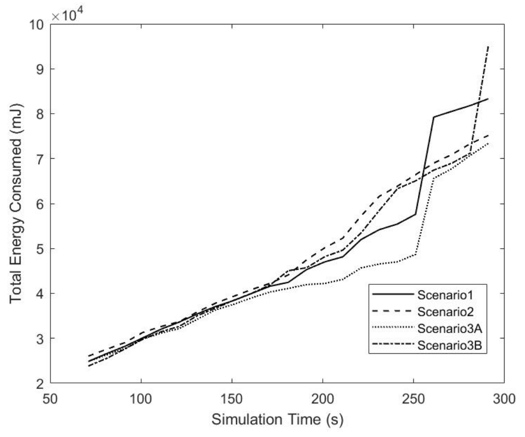
E_{elect} represents the losses in the electronics of the robot. A robot system includes DC motors driver, sensors and micro-controllers that compose the electronics of the robot. These components are also consuming part of the battery's energy. The equation is given below.

$$E_{elect} = \int (I_{elec}V_{elec})dt.$$

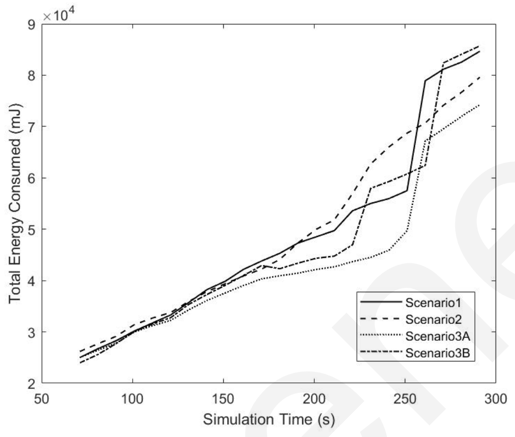
Energy Evaluation Results

In this part of the evaluation, our novel energy efficient solution is compared with different energy models based on different mobile robots characteristics.We use existing energy models based on three different types of mobile robots and simulate them with our own algorithm to compare their results. The main goal is to examine if the energy model is important in the energy consumption of the nodes in the network or the algorithm is the main resource of consumption.

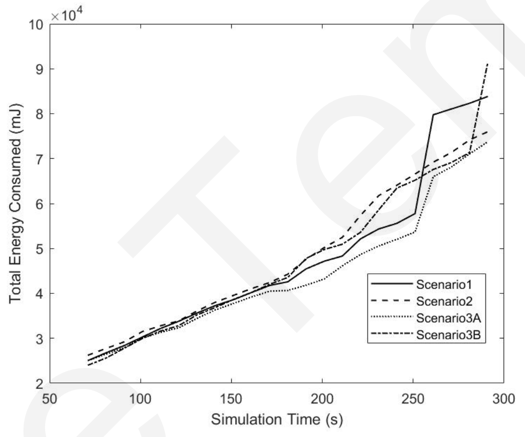
Evaluating the Energy Models. In this section we compare the different scenarios of the algorithm to each energy model.



(a) Energy Model 1



(b) Energy Model 2



(c) Energy Model 3

Figure 6.13: Evaluation of the different Energy Models

In Figure 6.13a we present the results for all scenarios using the Energy Model 1. At first all scenarios have similar results, due to the algorithms structure, and when the mobile node starts acting differently, each scenario has a different result. We can observe that Scenario 3B finishes with the most consumed energy, which is expected because it is the only scenario using the most mobile nodes.

In Figure 6.13b we present the results for all scenarios using the Energy Model 2. Scenario 3B is the one with the most consumed energy, following by Scenario 1 having that have a slightly difference. Scenario 2 has the least amount of consumed energy in the network, due to the fact that only one mobile node is used in the network.

In Figure 6.13c we present the results for all scenarios using the Energy Model 3. The same as the previous one, it is shown that Scenario 3B has the most consumption. The least consumption is seen in Scenario 2 where the mobile node is reused, which is normal as only one mobile node is needed in the network.

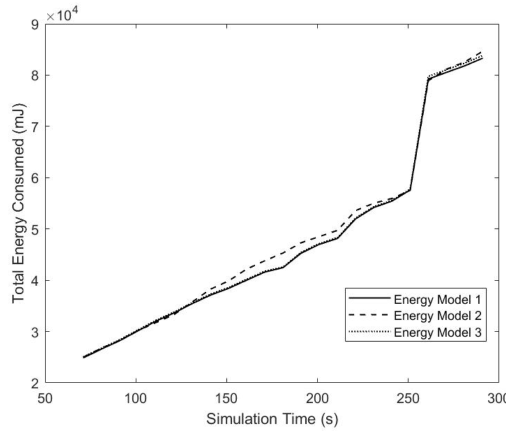
In general, we can observe that all energy models have the same performance while running our algorithm. Based on the results, it is shown that Energy Model 3 presents the largest energy consumption and Energy Model 1 has the least energy consumption. Based on the scenarios, it is shown that Scenario 3B is the one that uses the most mobile nodes in the network and takes the first place in energy consumption. Scenario 2 uses only one mobile node, because of reusing the mobile nodes in the network, and results in consuming the least energy of them all.

Evaluating the Energy Model based on the Scenario. In this section we compared different energy models for each scenario of the algorithm, to present the differences of each robot's characteristic. Does an energy model of a different robot have a different result in the total energy consumption of the network? It is worth mentioning that the speed of the node is set as 0.65m/s which is the same constant in all three energy models.

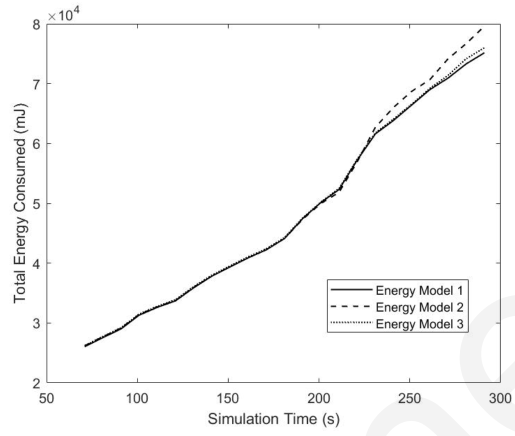
In Figure 6.14a we present the results of each energy model for Scenario 1. In this scenario the mobile nodes in the network at the end of the simulation are two. We can observe that Energy Model 2 and Energy Model 3 are the ones with a slightly small difference due to their equation that uses the time variable, whereas the Energy Model 1 uses the distance variable. All models use the same speed variable, so the difference relays in the other variables of their equation.

In Figure 6.14b we present the results of each energy model for Scenario 2. In this scenario, we reuse the mobile node in the network, so only one mobile node is inserted in it. In respect to the general energy consumption, the numbers are lower than the previous scenario, which is normal due to the number of mobile nodes used. In respect of the energy models used, the one using the distance (Energy Model 1) has higher consumption than the others using the time, which is normal due to the fact that the mobile node moves and its distance is changed, whereas the time changes constantly.

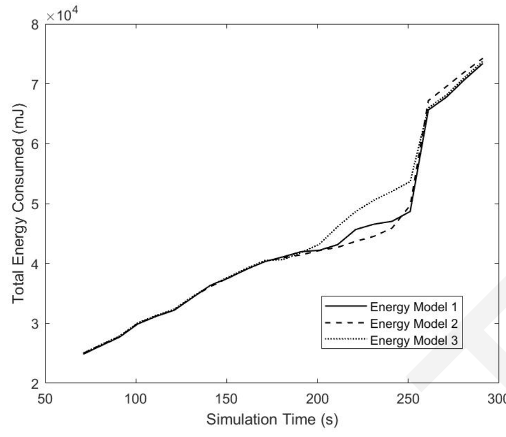
In Figure 6.14c we present the results of each energy model for scenario 3A. In this scenario, the mobile node in the network returns to its initial position due to energy lack and a new mobile node takes the second congestion. It is shown that until the mobile node leaves all models have the same reaction, whereas when the mobile node leaves and until the new one is placed to its position the energy does not change much. The changes in the consumption is due to the energy from the source node that are sending their packets and



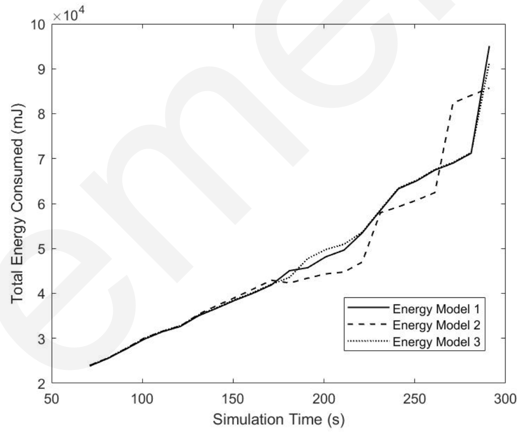
(a) Energy Consumption of Scenario 1



(b) Energy Consumption of Scenario 2



(c) Energy Consumption of Scenario 3A



(d) Energy Consumption of Scenario 3B

Figure 6.14: Evaluation of the different Scenarios

their relay nodes. When the new mobile node inserts the network the energy consumption increases faster than before.

In Figure 6.14d we present the results of each energy model for Scenario 3B. In this scenario, the mobile node returns back to its initial position due to energy lack but it will be replaced by another mobile node, because it is needed in the network. When the second congestion occurs, a new mobile node is inserted in the network. As a result, this scenario has the largest number of mobile nodes used and the most energy consumption. All models start to differ when the replacement is done and the source nodes send more packets in order to create the second congestion. Before the end, a new mobile node is entered in the network and the consumption increases. The highest number is returned by the Energy Models 2 and 3 that uses the time, whereas Energy Model 1 uses the distance consumed the least energy, because the distance does not change so often as the time.

In general, we can observe that Energy Model 1 consumes the least energy in the network, because the distance variable is not so frequently changed as the time variable. The two other energy models, 2 and 3, that use time, which changes at all times, have more consumption. The speed variable, which is a constant variable, does not really affect the result.

Discussion

The evaluation of our algorithm based on different energy model was to observe the impact of the model to the total energy. The results show that the algorithm has greater impact on the energy consumption of the node than the energy model. All energy models used in the evaluation provided similar results with a small difference depending on specific details. This is expected, since the energy models are based on same general criteria that relates to speed, time and distance. As a result, we can observe that the energy consumption of the nodes depends mainly on the algorithm and the steps they follow, rather on the specific energy model used.

6.5 Evaluation of cNPA

In this section, we present the evaluation scenarios and results for the Algorithm cNPA (Chapter 5). Wismote is another mote type of COOJA with better specifications than the Tmote Sky. Since this algorithm is more demanding in terms of memory and computational power, the new category was selected.

6.5.1 Evaluation Scenarios

Initially, we employed 32 Wismote nodes (1 sink, 19 fixed, 6 mobile nodes and 6 carriers) according to the topology of Figure 6.15.

In this scenario there are 1 sink node (node 1), 10 relay nodes (nodes 2-11, light grey), 9 source nodes (nodes 12-20, white), 6 mobile nodes (nodes 21-26, dark grey) and 6 carrier nodes (nodes 27-32, darker grey). The mobile nodes and carrier nodes are initially placed near the sink in a sleep mode until needed.

The first congestion occurs at node 3 and each approach has a different solution. In the Stay Approach (Fig. 6.16a), both mobile node 21 and carrier 27 move to the calculated position from the sink node, whereas in the Leave Approach (Fig. 6.16b), the carrier 27 takes the mobile node 21 to its new position and then returns to its initial position near the sink.

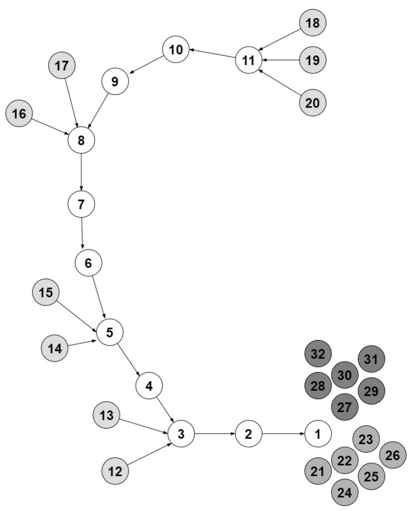
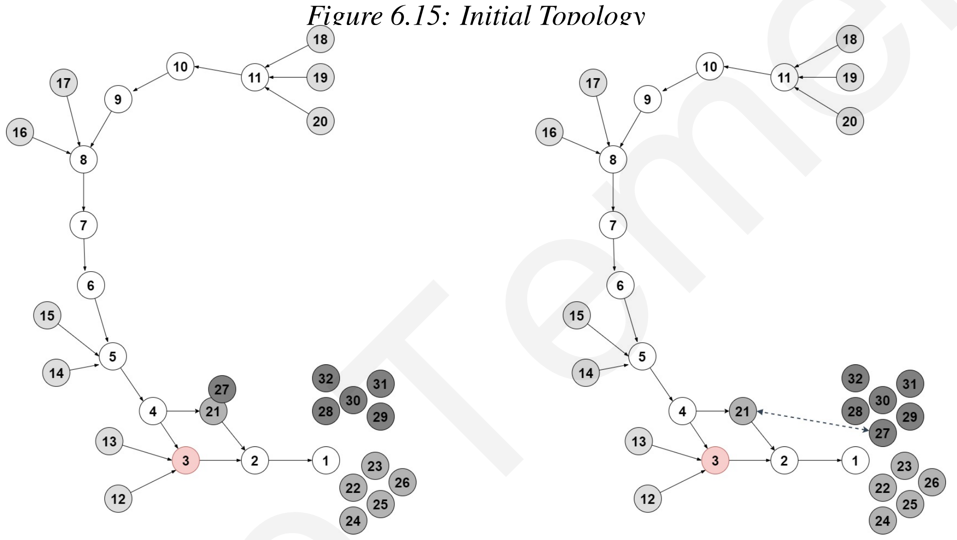


Figure 6.15: Initial Topology



(a) Stay Approach (b) Leave Approach

Figure 6.16: The first Congestion at node 3

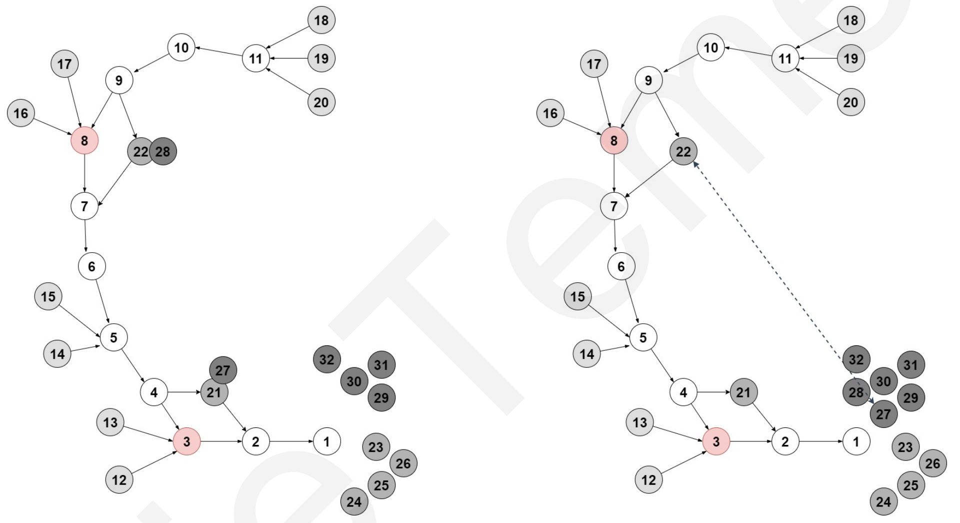
When a new congestion occurs at node 8, based on the factors of the mobile node already placed and the approach, the following scenarios are presented:

- Scenario 1: it presents a no-reuse scenario, where the mobile nodes are not used in a new problem, even if the old problem is resolved.
- Scenario 2: it presents a reuse scenario, where the mobile nodes are able to be reused for a new problem, if the conditions are applied to the node.
- Scenario 3: it presents a replacement scenario, where the mobile node reaches its energy exhaustion threshold and needs to return for charging, but it is still needed.

All scenarios presented use the two approaches of the mobile carriers. As a result, each scenario has two cases: (1) the Leave Approach, where the mobile carrier returns back after completing a task, and (2) the Stay Approach, where the mobile carrier is assigned to a mobile node.

Scenario 1

This scenario presents a no reuse solution, where the mobile node positioned for the first congestion problem is not available for reuse in another problem. When the second congestion occurs, a new pair of mobile node and carrier are moved to the new position calculated by the sink for the Stay Approach (Fig. 6.17a). However, in the Leave Approach (Fig. 6.17b), a carrier and a mobile node are selected, and the carrier after the placement of the mobile node returns to its initial position.



(a) Stay Approach *(b) Leave Approach*

Figure 6.17: Scenario 1 - No reuse

Scenario 2

This scenario presents a reuse solution, where the mobile node positioned for the first congestion problem is available for reuse in another problem. When the second congestion occurs, in the *Stay Approach* (Fig. 6.18a), the pair of mobile node and carrier move to the new position calculated by the sink node. However, in the *Leave Approach* (Fig. 6.18b), the carrier starts from its initial position, goes to the current position of the mobile node, takes the mobile node and move together to the new position. From there the carrier returns to its initial position.

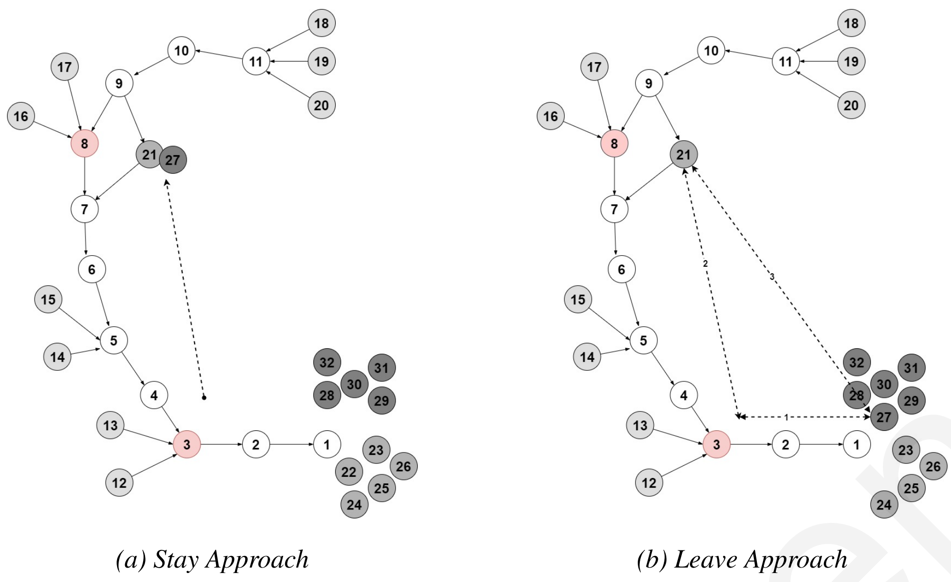


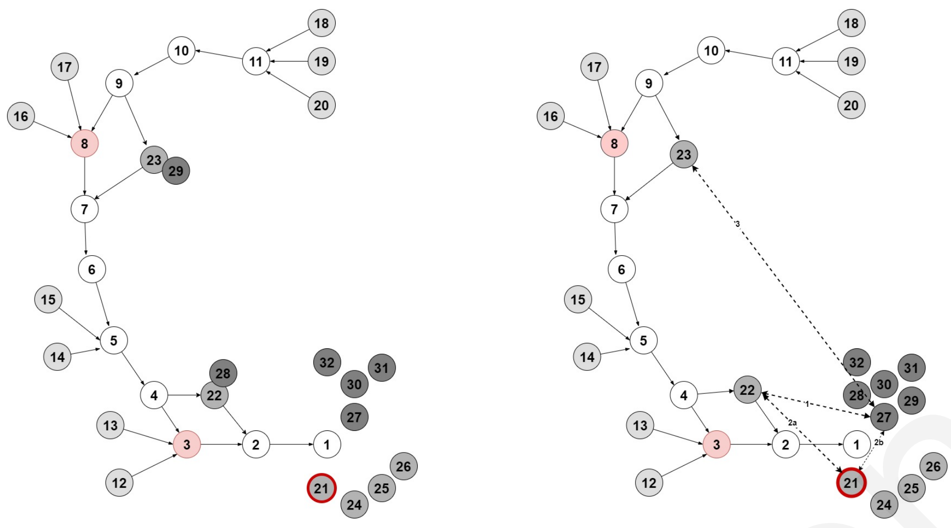
Figure 6.18: Scenario 2 - Reuse

Scenario 3

This scenario presents a replacement solution, where the mobile node positioned for the first congestion problem needs a replacement, as its battery has reached its lowest threshold. When the sink node is aware of this request a new mobile node is placed at this position. In the *Stay Approach* (Fig. 6.19a), a new pair of mobile node and carrier move to the current position of the requesting node, and at the same time the old pair returns to their initial position, near the sink to charge their battery. When the new congestion occurs, a new pair is sent to the new position. In the *Leave Approach* (Fig. 6.19b), the replacement is performed by the same carrier, which takes the new mobile node to the position and from their takes the old mobile node back to its initial position. When the new congestion occurs, the carrier takes a new mobile node to the new position and then returns back.

6.5.2 Evaluation Results

We now present some evaluation results over the mentioned scenarios. The proposed algorithm is evaluated in terms of network performance and algorithm performance. The network performance is evaluated through the percentage of successfully received packets and delay metrics and the algorithmic performance through the total energy consumption of the network. The graphs show the results from the time that the different scenarios start to run, since before that all tasks were the same with the same results. The starting time varies based on the metrics.



(a) Stay Approach (b) Leave Approach

Figure 6.19: Scenario 3 - Replace

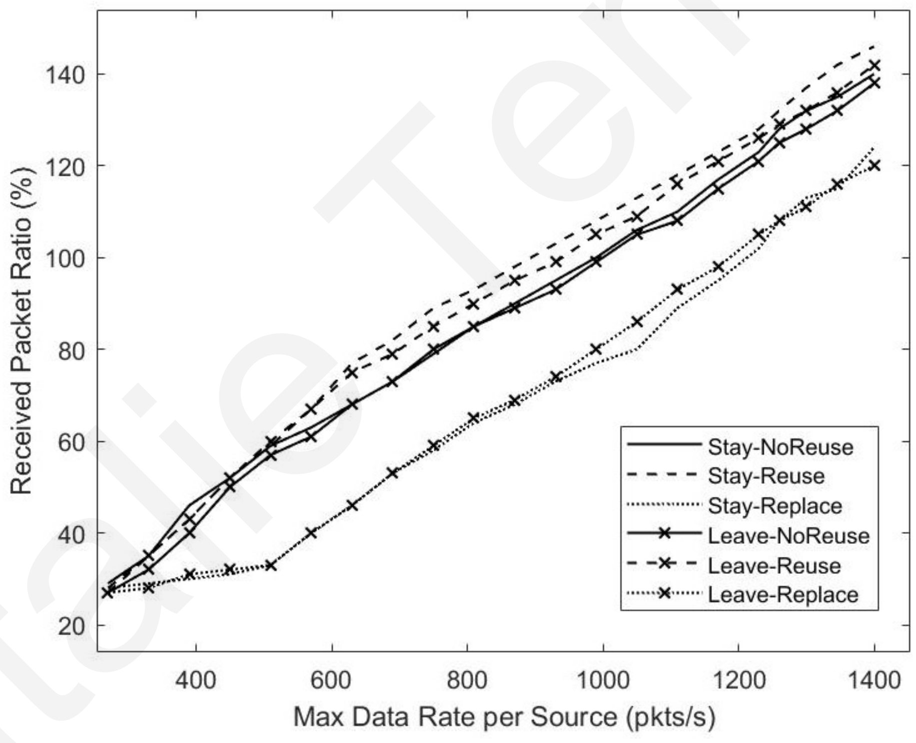


Figure 6.20: Percentage of Successfully Received Packets

In Fig. 6.20, the percentage of successfully received packets is presented. In this plot, we observe that the results of the *Stay Approach* are in all scenarios slightly better than the *Leave Approach*. This is normal because in the *Stay Approach*, the carrier is always with the mobile node, whereas in the *Leave Approach*, the carrier needs to find the mobile node and then continue with the actual action that needs to be done. As expected, the reuse scenario in both approaches has the best result, because the time needed for an in-use mobile node to be relocated is lower than the time needed to take a new one that is further away. Additionally, the *replace scenario* in both approaches presents the lowest results, which is normal as the time needed for replacement will have impact on lost packets.

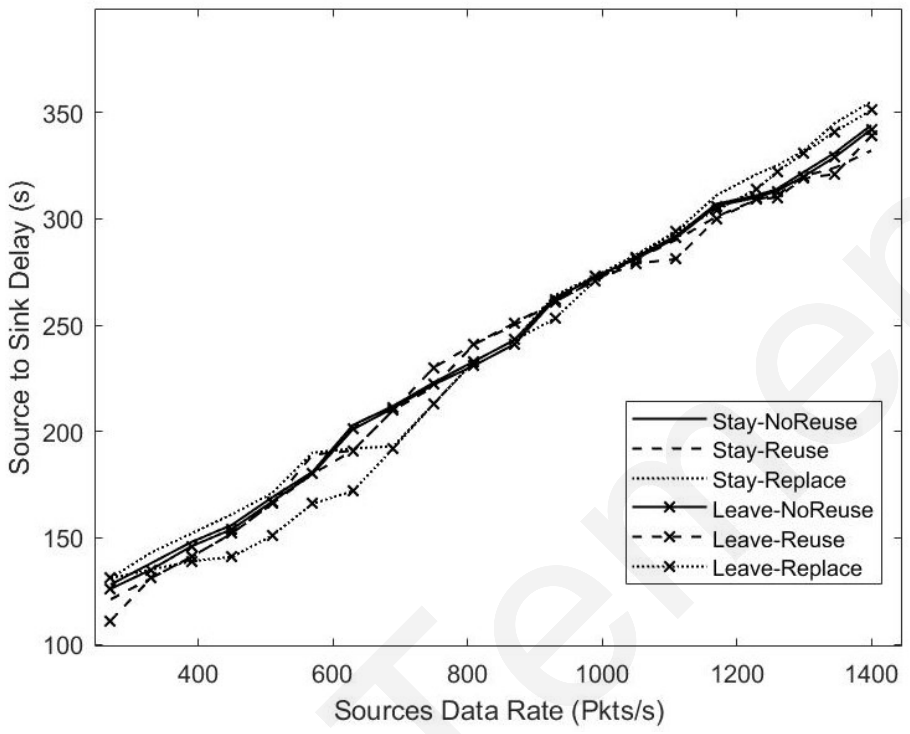


Figure 6.21: Source to Sink Delay

Fig. 6.21 presents the total source to sink delay of the network. The plot shows that all scenarios during the simulation seem to have an increased delay due to the existence of collisions, where many packets are retransmitted or sometimes dropped. It is observed that the scenario with the most delay is the *replace scenario*, which is expected because in this scenario more information is needed to be injected for accomplishing the task. On the other hand, the *reuse scenario* is the scenario with the lowest delay, because it has the fewer information requirements to complete the task, which means that fewer packets are dropped as the problems occurring are solved in a shorter time.

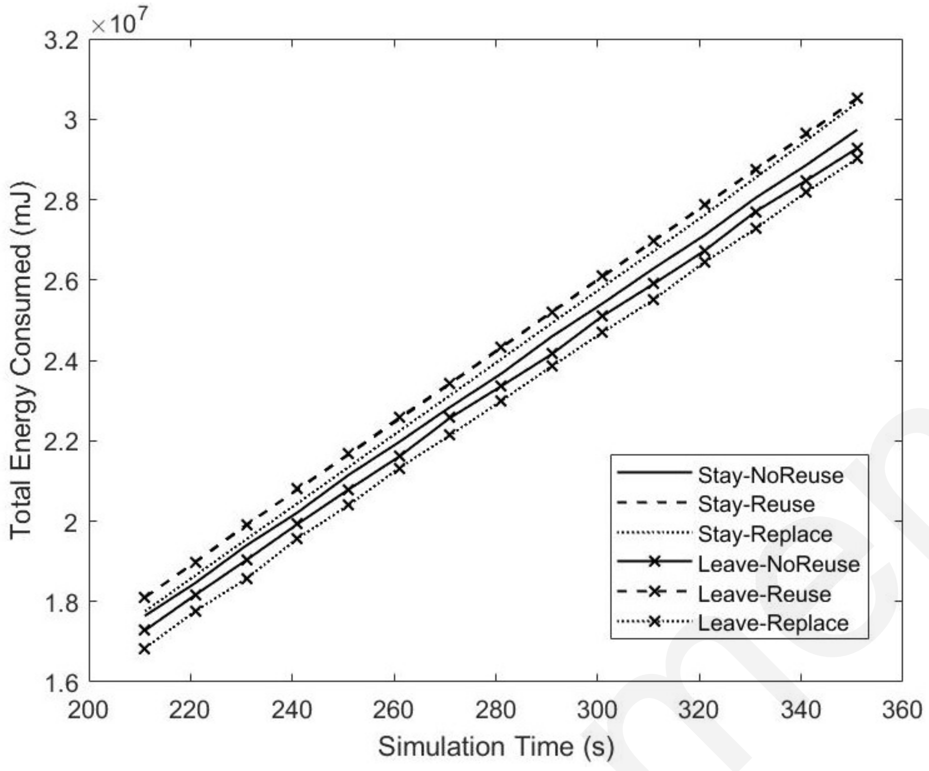


Figure 6.22: Total Energy Consumed

Fig. 6.22 shows the total energy consumption of the network. This plot shows the different energy consumption of each scenario. It is observed that for the replace and no reuse scenario the *Leave Approach* has better results than the *Stay Approach*, whereas in the *reuse scenario* the Stay Approach has slightly better results than the *Leave Approach*. This is normal as the Leave Approach although it uses fewer carriers, the distances of the carrier to move are bigger than the Stay Approach. The replace scenario of the Leave Approach uses the lowest energy consumption of all scenarios, because it only uses one carrier to make all actions needed, and the replacement is done at the same time, whereas in the Stay Approach this scenario needs two carriers and one for the new problem occurrence.

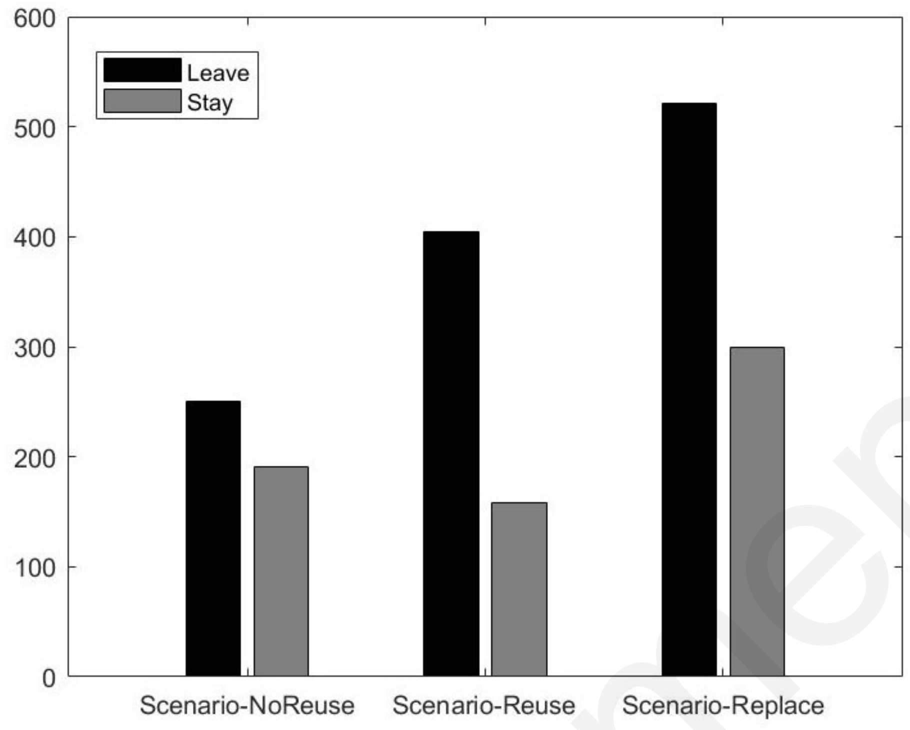


Figure 6.23: Total Distance

Fig. 6.23 presents the total distance of the carrier traveled in the network during the whole simulation. In the plot, it is shown that the carriers used in the *Leave Approach* travel more distance than the ones in the *Stay Approach*. The results are normal because in the Leave Approach a carrier always returns to its initial position so the additional last distance is not included in the carriers of the Stay Approach. The *noreuse scenario* has the smallest difference between the approaches for all scenarios, since in both approaches the same amount of carriers is used. The *replace scenario* has the biggest distance, since in this scenario an extra task is performed, the replacement, which adds extra distance than the other two scenarios. The *reuse scenario* has the best results in the Stay Approach as only one carrier is needed, whereas in the Leave Approach, the carrier needs to travel to the first position, then to the destination and return back.

Discussion

The evaluation results show that using a carrier for the transportation task, mobile nodes are able to use the fullest of their energy for accomplishing their task without considering the energy for moving from and to the locations. The Stay Approach uses more resources as each mobile nodes has a carrier assigned, but it does not have the overhead of the Leave Approach for returning back to the initial position and the first step of finding the position of an in-used mobile node. This overhead is the main reason of many delays in the Leave Approach. In general both approaches, stay and leave, show promising results for different environments. The Stay Approach can be used in environments where the carrier needs to relocate the nodes frequently. On the other hand, the Leave Approach is a good solution for hostile environments where it is safer to return to its initial position.

6.6 Comparison between NPA, eNPA and cNPA

Based on the evaluation results of each algorithm (NPA, eNPA and cNPA), it is observed that the use of mobile nodes can alleviate the problem that has occurred in the network. As each algorithm is a better version of the previously designed, it is important to compare them with each other to analyze the advantages and disadvantages of them. In general, Algorithm NPA is the first algorithm implemented and has the simplest design, as a result each one of the other two algorithms (eNPA and cNPA) can outperform it.

In terms of energy consumption, Algorithm cNPA uses the mobile nodes to their fullest energy, as the addition of carriers removes the responsibility of transportation to each point from the mobile node. However, this algorithm exhibits the largest total energy consumption, which is expected as extra resources are employed in the network, meaning the carrier. These additional resources in total are extra nodes that the other two do not have in their network, so in general cNPA uses more resources. Algorithm eNPA, which introduces the term of reuse by adding the energy metric to the calculations, is able to analyze and use the energy of the mobile nodes in a more efficient way. Whereas, Algorithm NPA does not take into consideration any parameters except the need to serve as many affected nodes as possible. It is worth mentioning that the Dynamic MobileCC variation uses less energy than the Direct Path MobileCC variation, as the latter creates an alternative path, which means it uses more mobile nodes for resolving the problem.

In terms of time, the simplicity of Algorithm NPA provides a faster computational mechanism which leads to resolving the problem faster. The use of an extra metric in the calculations performed by the sink node, meaning the energy consumption provided by Algorithm eNPA, takes more time for computations, which leads to a little slower solution than Algorithm NPA. Additionally, the addition of carriers in cNPA, adds a lot more time to the computations performed by the sink node. This lead in needing more time for calculating the new position of a mobile node in the network. However, in all three algorithms, the additional time needed is not a big overhead in the total time needed to resolve the problem and restore the network to a stable state.

Essentially, each algorithm adds an extra feature to the previously implemented algorithm, which makes it an improved version in terms of performance. These extra features provide a more detailed analysis of a parameter that take into consideration a new fact about the solution. The new fact is able to provide a more detailed and analyzed variation that is more realistic. The main goal is to create a solution that can increase the lifetime of the network by employing mobile nodes and using them to their fullest potentials.

Chapter 7

The Fault Tolerant Node Placement Algorithm

Faults is one of the most important and vital challenges that need to be addressed, as the limitations of IoT networks and WSNs make them vulnerable to different types of faults.

Fault tolerance [9] is the technique that refers to the capability of a system to deal with faults and to maintain its functionality. One of the most common approaches used to increase the Fault Tolerance of a network is Fault Management [53]. A Fault Management Framework consists of three steps: fault detection, fault diagnosis and fault recovery.

The fault detection step is responsible for detecting any factor that impacts the network or a node. This technique can be divided into three approaches: *centralized, distributed* and *self-managed*. In the first approach, a centralized node, like the sink, is responsible to monitor the network and detect a faulty node. In the second approach, the detection is handled by all nodes using the neighboring nodes and clustering methods. In the third approach, a node is responsible to investigate and analyze its own needs and then report.

The fault diagnosis step is the step that identifies the fault, in terms of the type, the cause, the effect on the network, etc. This step is further divided into four main monitoring approaches: *passive, active, proactive* and *reactive*. In passive monitoring, the diagnosis of a fault activates alarms, whereas, in active monitoring, sensor nodes are instructed to report their existence to a control center node (e.g., sink) with alive messages. In proactive monitoring, every past diagnosis is analyzed and future events are predicted in order to maintain the performance of the network. Reactive monitoring is defined as a management system that gathers information about the network state for detecting interesting past events in order to take specific adaptive measures to reconfigure the network.

The last step, called recovery, is responsible to reconfigure or rebuild the network so that faulty nodes can no longer affect the network's functionality and performance. This essentially means that the faulty state of the network is replaced by a fully functional state.

This technique is further divided into *recovery* and *reconfiguration*. The former removes the impact of the fault and the latter changes the structure of the network without affecting the overall output.

7.1 Mobile Fault Tolerant Framework

We consider a network that consists of static nodes that are randomly deployed and a small set of mobile nodes that are initially placed near the sink node.

The primary objective of this work is to utilize the extra resources, mobile nodes, effectively and efficiently in order to resolve faults in the network and, if possible, to improve the performance.

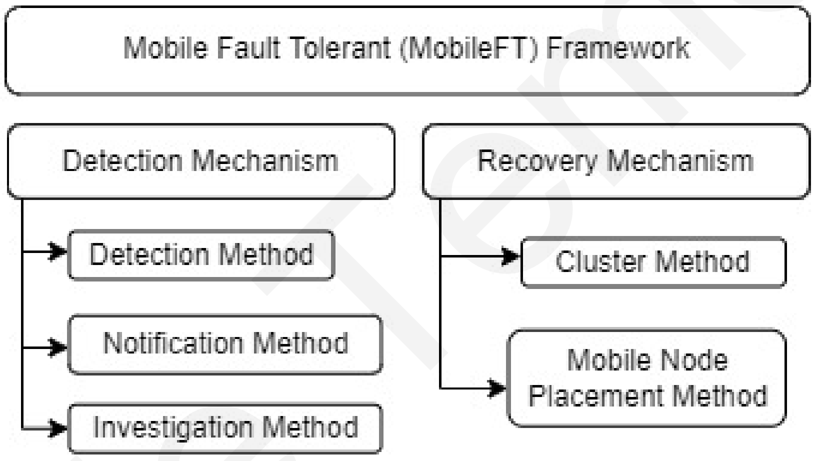


Figure 7.1: Diagram of the MobileFT Framework

The Mobile Fault Tolerant (MobileFT) Framework consists of the following mechanisms (see Fig. 7.1):

- Fault Detection mechanism
  - Detection method
  - Reporting method
  - Investigation method
- Fault Recovery mechanism
  - Cluster Method
  - Mobile Node Placement method

The fault detection mechanism consists of three methods: *detection*, *reporting* and *investigation*. In the detection method, the faulty node is detected in the network and in the reporting method, the sink node is notified about the fault that has been detected. In the investigation method, the sink node sends a mobile node to investigate the affected area and report back to it.

The fault recovery mechanism consists of two methods: *cluster method* and *mobile node placement method*. In the cluster method, the results from the investigation method are processed and the node that are identified as faulty are divided into clusters. In the mobile node placement method, the mobile nodes are placed in the network in positioned calculated by the sink node in order to assist the affected area(s).

7.2 The Fault Tolerant Node Placement Algorithm

The Fault Tolerant Node Placement Algorithm (FTNPA) consists of two variations, decentralized and centralized, based on the detection method used. The Decentralized FTNPA algorithm detects the fault with the use of the neighboring nodes, whereas in the Centralized FTNPA algorithm, the sink node is responsible to detect any faults that occur in the network. We further describe in detail both variations.

7.2.1 Decentralized Fault Tolerant Node Placement Algorithm

The Decentralized Fault Tolerant Node Placement Algorithm (DFTNPA) is divided into two mechanisms: the detection and the recovery. The former detects the faults in the network, while the latter uses mobile nodes to recover the network and stabilize it. Algorithm 16 describes the functions of the sink node and Algorithm 17 describes the functions of the mobile node. We clarify that the detection mechanism considered in this work involves non-malicious type of faults, such as crashes.

Detection Process

Every static node in the network is able to identify the following faults:

- Energy level (battery exhaustion): If the remaining energy is becoming lower than a given threshold, then the node immediately informs the sink node about the upcoming (self)node failure.

Algorithm 16 The DFTNPA algorithm for sink node s

```
 1: upon receive ("Fault Notification($fn$)") from $n_i$ then
 2: pos = position of $fn$
 3: expectedNeighborList = all nodes in the range of $fn$
 4: select mobile node $m_i$
 5: send(Investigation Notification(pos, level, expectedNeighbors)) to $m_i$
 6: upon receive ("Investigation Notification Response(aliveNodesList)") from $m_i$ then
 7: for all $n_j \in expectedNeighborList$ do
 8:     if $n_j \notin aliveNodesList$ then
 9:         add $n_j$ in pathList
10:     end if
11: end for
12: add $pos_{m_i}$ in pathList
13: send(Investigation Path(pathList)) to $m_i$
14: upon receive ("Investigation Path Response(nodesList)") from $m_i$ then
15: create clusters for all $n_j \in nodesList$
16: for all $c_i \in clusters$ do
17:     pos = DynamicFT($c_i$)
18:     select mobile node $m_j$
19:     send(New Position(pos)) to $m_j$
20: end for
```

- Faulty neighboring node: The node can identify a faulty node from its neighboring list when a certain amount of time has passed from the last time of their communication.

The node, upon detecting one of the above faults, attempts to send a *Faulty Notification Message* (FNM) to the sink, which contains all the information needed, such as the node ID and its level in the network. When the fault is referred to energy exhaustion, the node sends its own information, whereas in the case of a neighboring failure, the node sends the neighbor's information, which can be found in its neighboring table.

Upon receiving an FNM message, the sink forms the set of (expected) neighboring nodes of the faulty node, and then sends an *Investigate Notification Message* (INM) to a selected mobile node. INM includes the target position, the set of expected neighboring nodes, and its new level (see Alg. 16, lines 1-5). Upon receiving the INM message, the mobile node moves to the target position and starts the investigation process.

Investigation Process: This process is divided into two steps: the *neighborhood investigation* and the *one-level investigation*. In the neighborhood investigation, the mobile node broadcasts an "alive" message to its neighborhood and waits for the nodes to respond (see Alg. 17, lines 1-10). When the timer expires, the mobile node sends an *Investigation Notification Response* (INR) message including all nodes that have responded to its message. The sink upon receiving the INR message finds which nodes did not respond to the message of the mobile node. The positions of these nodes are sent to the mobile node with the *Investi-*

Algorithm 17 The DFTNPA algorithm for mobile node m_i

```
1: upon receive ("Investigation Notification(pos, level, expectedNeighbors)") from sink then
2: move to pos
3: broadcast an "alive" message
4: wait for responses until timer $T_r$ expires
5: if reponse received then
6:     add $n_j \in aliveList$
7: end if
8: if $T_r$ expires then
9:     send(Investigation Notification Response(aliveList)) to sink
10: end if
11: upon receive ("Investigation Path(pathList)") from sink then
12: for all $p_i \in pathList$ do
13:     move to $p_i$
14:     if $p_i$ == start_point then
15:         send(Investigation Path Response(aliveList)) to sink
16:         break
17:     end if
18:     broadcast alive message
19:     wait for responses until timer $T_r$ expires
20:     if reponse received from $n_j$ then
21:         add $n_j \in aliveList$
22:     end if
23:     if $T_r$ expires then
24:         move to next $p_i$
25:     end if
26: end for
27: upon receive ("New Position(pos)") from sink then
28: move to pos
```

gation Path (IP) message (see Alg. 16, lines 6-12). In the one-level investigation, the mobile node receives from the sink the IP message that contains the positions of the non-responding nodes that it should visit and investigate. At each position, the mobile node broadcasts an “alive” message and waits for the responses. At the end of this procedure, the mobile node returns to its starting point and informs the sink node about its finding with an *Investigation Path Response* (IPR), which includes all the nodes it found alive without an upper node (i.e., the nodes that are alive but due to neighboring node faults, did not have a path to the sink); see Alg. 17, lines 11-26.

Recovery Process

Based on the information collected by the IPR message, the sink calculates the number of mobile nodes and their positions that are needed for restoring the network. This procedure is divided into two methods: the *clustering method* and the *positioning method* (see Alg. 16, lines 13-19). Firstly, the constraint clustering method (cf. Section 7.2.1) is used, where the sink node divides all nodes without an upper node into clusters based on their positions with a variation of the k -means clustering algorithm [44]. Then, the positioning method is used, where the Dynamic MobileFT algorithm (cf. Section 7.2.1) runs to find the position of the mobile node for each cluster created and is described below. When the position is calculated, the sink node selects an available mobile node and sends it to the calculated position. When a mobile node receives a *New Position* message it moves to the position and starts acting like a static node (see Alg. 17, lines 27-28).

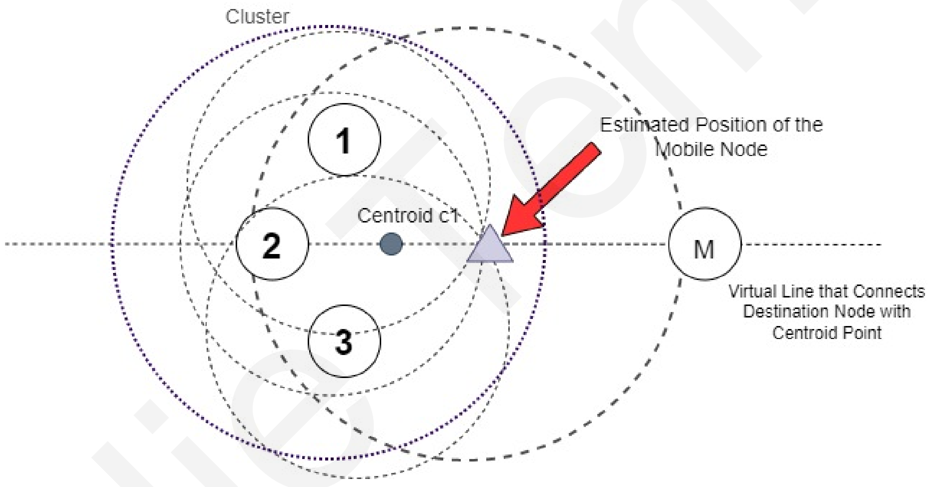
Constraint Clustering Method. For the clustering method, a new variations of the k -means clustering algorithm [44] has been developed. The clustering method uses an unsupervised machine learning technique. This means that the process is not able to be guided or supervised by any external knowledge. The main goal is the creation of subgroups so that each data point in the same cluster are similar and the ones in different clusters are dissimilar.

One of the many clustering approaches used is the partitioning clustering. In this approach the method for creating the clusters uses some parameters, such as the number of clusters to be created, the distance measurement and the points that need to be assigned to the clusters. One of the most known partitioning clustering algorithms is the k-means method due to its simple implementation. Therefore we decided to use this algorithm for our purposes. Considering different clustering algorithms is an interesting future research direction. The k-means algorithm [44] is an iterative technique, where its goal is to partition a specific dataset into k separate non-overlapping clusters. To do so, the algorithm uses distance-based measurements for determining the similarities between the data points. The cluster representatives are defined as the centroids calculated by the algorithm.

In order to extend the classic clustering algorithms with the use of existing domain knowledge, the constrained clustering [83] was introduced. This clustering method uses a semisupervised learning technique, which uses the knowledge as a general rule to constraints. The difference from an unsupervised algorithm is that, except the parameters mentioned before, the input of this algorithm uses also a set of constraints. These constraints can be either enforced in the solution, as hard requirements, or used as guidance.

To meet our needs, we created a new variation of the classic k-means clustering algorithm by adding some constraints. The constraints consists of two groups: (a) as a hard requirement and (b) as guidance. The *constraint as a hard requirement* is responsible for the coverage range of the cluster centroid. The range of a cluster is defines as the communication range of a node. As a result, all cluster members need to have less than this distance from the centroid point of their cluster. The *constraint as guidance* is responsible to speed up the process of clustering. The idea is that every time a node is without a cluster, which means that the distance between the centroid is greater than the range, it will be assigned to an empty cluster and all clusters are then rearranged. At the end of the clustering algorithm, all clusters are assigned to a cluster, and each cluster member has a distance less than its range from the cluster's centroid point.

Dynamic MobileFT. The main idea of this algorithm is to find the position of the mobile node that can serve all target nodes. The target nodes are all part of a cluster, where the centroid point of this cluster is known. Based on this centroid point, the algorithm creates a line from the centroid towards the destination point, the firstly placed mobile node, and find the best position that can serve all target nodes. This is illustrated in Fig. 7.2.



Cluster
1
Estimated Position of the Mobile Node
2
Centroid c1
M
3
Virtual Line that Connects Destination Node with Centroid Point

Figure 7.2: Dynamic MobileFT

An example of placing a mobile node based on Fig. 7.2 is described below. Mobile node M is the firstly placed mobile node that was assigned to replace the first faulty node and was responsible for the investigation process. This node is the destination node where the disconnected nodes of the cluster need to connect to. Nodes 1, 2 and 3 are all assigned to a cluster with centroid c. Then, the line is created that passes from point c (pos_c) and from the position of M (pos_M). For each node in the cluster, the intersection point with the line is calculated. This point is then checked on whether is in range of the other cluster members. If this is true, then the point is selected to place a new mobile node, otherwise the centroid point is selected.

Firstly, the line created from the centroid of the cluster and the destination node (firstly placed mobile node) is calculated (see Alg. 18, line 9). Then, for each node in the *targetList*,

Algorithm 18 Dynamic MobileFT algorithm

```
1: line = findLine(centroid,s)
2: for all $n_i \in targetList$ do
3:     $pos$ = intersectionPointofLine&Circle($line$,$n_i$)
4:     for all $n_j \in targetList$ do
5:         $flag_{range}$ = isInRange($pos_{n_j}$,$pos_{m_i}$)
6:         if $flag_{range} == true$ then
7:             count++
8:         end if
9:     end for
10:    if $count == targetList.size()$ then
11:        return pos
12:    end if
13: end for
14: return centroid
```

the intersection point of the node and the line is calculated (see Alg. 18, lines 10-11). The point chosen is the one which is closest to the first placed mobile node for this problem. This point is then checked with the function *isInRange* that checks if the point is in the range of this mobile node. When all nodes are covered from a point, this point is returned, otherwise the centroid point is used (see Alg. 18, lines 12-22).

7.2.2 Centralized Fault Tolerant Node Placement Algorithm

The Centralized Fault Tolerant Node Placement Algorithm (CFTNPA) is divided into two mechanisms: the detection and the recovery. The former is responsible for detecting the faults that occur in the network, while the latter is responsible to recover and stabilize the network with the use of mobile nodes.

Detection Method

The sink node is able to identify a fault when it stops receiving packets from a certain node in the network (see Alg. 19, line 38). Based on a periodic check function that identifies the last time the sink has received a message from a node in the network (see Alg. 19, lines 1-12), the sink can detect if a fault has occurred. When such a fault is recognized, the sink creates the Faulty Node (FN) list (see Alg. 19, line 40). This list is created based on the nodes that may be faulty, meaning the nodes that the sink has stopped receiving packets from (see Alg. 19, lines 13-20). Based on these nodes, the Suspicious Faulty Nodes (SFN) List is created (see Alg. 19, line 41). This list contains all nodes that are in the area of the nodes in the *FNList*. Initially, the center of the area that covers all FNList nodes is found. Based on this center point, the area of the SFN nodes is created, which is defined as double the size of the range

Algorithm 19 The CFTNPA algorithm for sink node s

```
1: function CHECKFORFAULTS(nodesList)
2:     for all $n_i \in nodesList$ do
3:         if $n_i.time_{received} >= Threshold$ then
4:             count++
5:         end if
6:     end for
7:     if $count == 0$ then
8:         return FALSE
9:     else
10:        return TRUE
11:    end if
12: end function
13: function CREATEFLIST(nodesList)
14:     for all $n_i \in nodesList$ do
15:         if $n_i.time_{received} >= Threshold$ then
16:             add $n_i \in FList$
17:         end if
18:     end for
19:     return $FList$
20: end function
21: function CREATESNFLIST(FList,nodesList)
22:     $c_F$ = findCenterofFaultArea($FList$)
23:     for all $n_i \in nodesList$ do
24:         $flag_R$ = inRangeofFaultyArea($pos_{n_i}$,$pos_{c_F}$,$R2$)
25:         if $flag_R == TRUE$ then
26:             add $n_i \in SNFList$
27:         end if
28:     end for
29:     return $SNFList$
30: end function
```

of a node. The SFNList is created and contains all nodes that are in the calculated area (see Alg. 19, lines 21-30).

Once the SFN list is created, the sink node divides the nodes into clusters based on their position (see Alg. 19, line 42) and then creates the path list, which contains all the centroids of each cluster (see Alg. 19, lines 31-37). A mobile node is chosen to be sent in order to investigate the clusters (see Alg. 19, line 43). The sink informs the selected mobile node with a *Discovery Path* message, which includes the *pathList* (see Alg. 19, line 44).

Upon receiving such a message, the mobile node moves to each position of the *pathList* from the sink to discover the alive nodes. At each position, the mobile node broadcasts an *alive* message and waits for responses (see Alg. 20, lines 1-9). When a node in the network receives such a message, it replies with an *Alive Reply* message that contains its delivery status, meaning if it has an active path towards the sink or not (see Alg. 21, lines 1-13). When the mobile node completes its journey, it returns to the sink node to inform it about its

```
31: function CREATEPATHLIST($clusters$)
32:     for all $c_i \in clusters$ do
33:         find centroid of cluster $c_i$
34:         add $pos_{centroid} \in pathList$
35:     end for
36:     return $pathList$
37: end function
38: $flag_f$ = checkForFaults($nodeList$)
39: if $flag_f == true$ then
40:     $FList$ = createFList($nodeList$)
41:     $SNFList$ = createSNFList($FList$,$nodeList$)
42:     $clusters_{SNF}$ = createClusters($SNFList$)
43:     $pathList$ = createPathList($clusters_{SNF}$)
44:     choose mobile node $m_i$
45:     send($Discovery\ Path(pathList)$) to $m_i$
46: end if

47: upon receive ("Discover Path Response($pathNodes$)") from $m_i$ then
48: for all $n_j \in pathNodeList$ do
49:     if $n_j.upper == FALSE$ then
50:         add $n_j$ in $cList$
51:     end if
52: end for
53: $clusters_F$ = createClusters($cList$)
54: for all $c_i \in clusters_F$ do
55:     run DirectPathMobileCFT($c_i$)
56: end for
```

findings (see Alg. 20, line 10).

Recovery Method

Based on the information the sink nodes receives, it will start the recovery process. The recovery mechanism consists of two methods: the cluster method and the mobile node positioning method. In the first method, all findings from the mobile node will be processed by the sink to find which nodes are without an upper node (see Alg. 19, lines 48-52). These nodes are used and divided into clusters. The clusters are based on the position of these nodes (see Alg. 19, line 53). In the second method, the new path of mobile nodes is created (see Alg. 19, lines 54-56) with the use of the Direct Path MobileFT algorithm (see Alg. 22), which is described below.

Constraint Clustering Method. For this method, a modified version of the classic k-means clustering algorithm [44] is employed that uses constraints, as mentioned in Section 7.2.1.

Algorithm 20 The CFTNPA algorithm for mobile node m_i

```
1: upon receive ("Discover Path(pathList)") from sink then
2: for all $pos \in pathList$ do
3:     move to pos
4:     broadcast an Alive message
5:     wait for responses
6:     if Alive Reply received then
7:         add $n_j \in pathNodes$
8:     end if
9: end for
10: send(Discover Path Response(pathNodes)) to sink
11: upon receive ("New Position(pos)") from sink then
12: move to pos
```

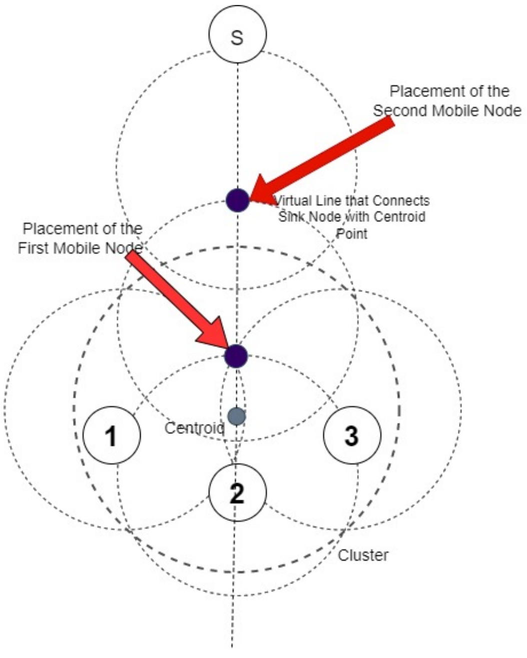
Algorithm 21 The CFTNPA Algorithm for node n_i

```
1: upon receive ("Alive Message()") from $m_i$ then
2: for all $n_j \in neighborList$ do
3:     if $n_j.hop < n_i.hop \&\& n_j.status == OK$ then
4:         counter++;
5:     end if
6: end for
7: if counter == 0 then
8:     send(Alive Reply(NO)) to $m_i$
9: else if counter > 0 then
10:     send(Alive Reply(YES)) to $m_i$
11: end if
```

Direct MobileFT. The main idea of this algorithm is to create a path of mobile nodes. For the first mobile node placed, the Dynamic MobileFT (see Alg. 18) algorithm is called to calculate its position. Then, a direct line is created that starts from the first placed mobile node and the destination. On the line, additional mobile nodes are placed in the range of the previous placed mobile node until one of them is in the range either of the sink node, which is the target destination node, or an alive node in the network where it can forward packets directly to it. This is illustrated in Fig. 7.3.

An example of creating a mobile node path based on Fig. 7.3 is described below. Once the first mobile node is placed with the use of the Dynamic MobileFT algorithm (cf. Section 7.2.1), the path starts to be developed. Each new mobile node is placed in the range of the previously placed mobile node and the procedure stops when the last placed mobile node is in the range of the destination node. The destination node of this path is either the sink node (see Fig. 7.3a) or an active node (see Fig. 7.3b) in the network, which needs to be placed on the virtual line created to connected the affected area and the sink node in order to be used as a destination node.

Initially, the Dynamic MobileFT algorithm is called, to get the position of the first mobile


S
Placement of the Second Mobile Node
Virtual Line that Connects Sink Node with Centroid Point
Placement of the First Mobile Node
Centroid
1
2
3
Cluster
(a) Path to the Sink Node

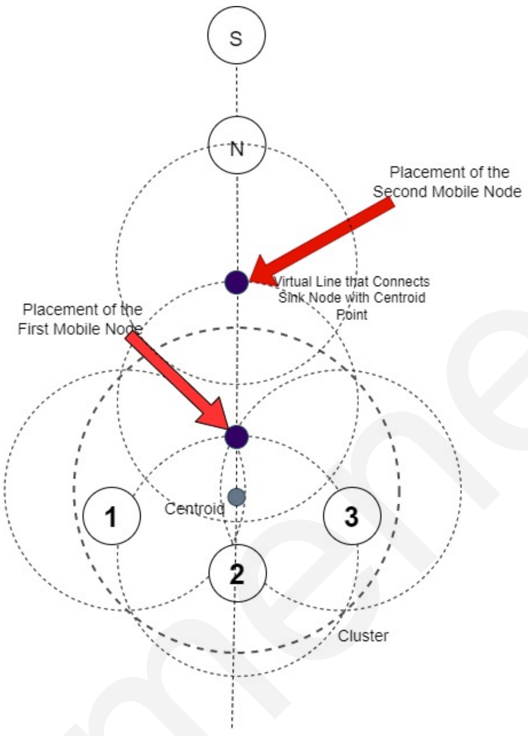

S
N
Placement of the Second Mobile Node
Virtual Line that Connects Sink Node with Centroid Point
Placement of the First Mobile Node
Centroid
1
2
3
Cluster
(b) Path to an Active Node

Figure 7.3: Direct Path MobileFT

Algorithm 22 Direct Path MobileFT algorithm

```
1: pos = DynamicMobileFT()
2: flagdp = false                                        ▷ a flag to stop the loop
3: while flagdp == false do
4:     choose $m_i \in mobileNodeList$
5:     send(New Position(pos)) to $m_i$
6:     isInRange = checkinRange(pos,$r$)
7:     if isInRange == true then
8:         flagdp = true;
9:     else
10:        pos = intersectionPofLine&Circle(pos,$nodeList$,$r$)
11:    end if
12: end while
```

nodes (see Alg. 22, line 1), which is sent to the calculated position (see Alg. 22, line 4-5). Then, this position is checked if it is in the range of the sink node or another active node in the network (see Alg. 22, line 6). When it is in the range, the mobile node is placed to its position and the procedure stops, otherwise, a new position is calculated in the range of the previously placed mobile node (see Alg. 22, lines 2-14).

7.3 Experimental Evaluation of FTNPA

To verify the effectiveness of our algorithms, we run two scenarios. The first scenario evaluates the Decentralized Fault Tolerant Algorithm (DFTNPA), as presented in Section 7.2.1, by comparing it to a benchmark solution of replacement. In the second scenario we evaluate the Centralized Fault Tolerant Algorithm (CFTNPA), as presented in Section 7.2.2, which is also compared to the benchmark solution.

7.3.1 Evaluation Setup

As in Chapter 6, we have implemented the algorithms within the Contiki OS [58], an open source operating system for implementing networked, resource-constrained systems, mainly focusing on low-power wireless Internet of Things devices. The evaluation has been performed in the COOJA simulator, a dedicated simulator for Contiki OS nodes.

Table 7.1 presents the simulation parameters. Most of the parameters used are the default parameters in the COOJA simulator, which are also mapped to the parameters of real sensor nodes. The total time of the simulation was selected to 30 minutes. This amount is enough for all two mechanism to run and also be able to run the network after reestablishing a non-faulty state. The simulation time has increased in respect to the previous evaluation, since this algorithm needs more time for the detection and recovery mechanism. The mobile node speed parameter is used in respect of the speed for the Wifibot mobile robot [96]. The maximum speed of a Wifibot mobile robot is 0.9 m/s, so the speed 0.65 m/s was selected that is not maximum but also not very small.

Table 7.1: Simulation Parameters

Simulator/OS	COOJA/Contiki 3.0
Protocol	Contiki Multihop/Rime
MAC	ContikiMAC/CSMA
Simulation Time	30 mins
Simulation Repetition	10 times
Emulated Mote	Tmote sky
Transmission Range (m)	25
Max Data Rate (kbps)	250
Queue Length (Pkts)	8
Packet Size (Bytes)	48
Mobile Node Speed	0.65 m/s

The network is set up and let to reach steady state for one minute. All sensor nodes are equivalent to Sky Mote nodes and have a 10m radio range. Each source node transmits one data packet of 48 bytes every 20 seconds. The faults start after the second minute of simulation. For each scenario 10 experiments were conducted each starting with a random seed.

To evaluate the FTNPA algorithm, the Replacement algorithm is used as a benchmark of comparison. This method uses the replacement technique, where each faulty node is replaced by a new mobile node. The replacement is performed once the sink is informed about the fault detection.

7.3.2 Evaluation Scenarios

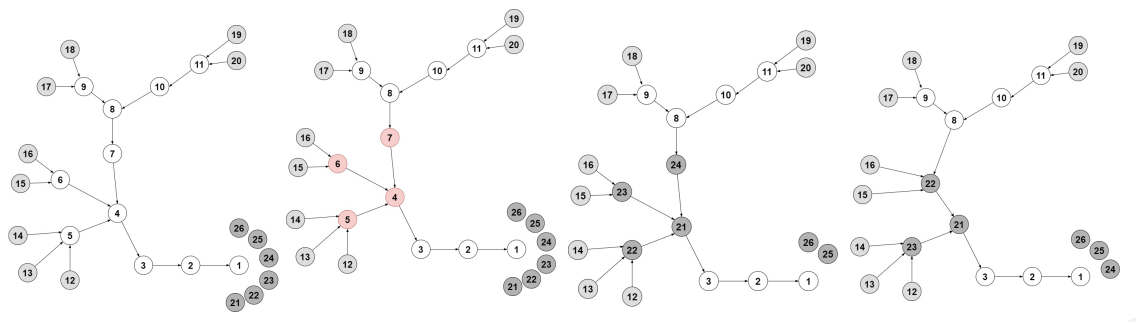
In this section, the evaluation scenarios for each version of the algorithm are presented. The scenarios used in the evaluation process are based on different topologies and different numbers of faulty nodes.

DFTNPA Scenarios

We implemented three different scenarios with different topologies and different number of faulty nodes. All scenarios employ 26 nodes, where 1 sink node (node 1), 10 relay nodes (nodes 2-11, white), 9 source nodes (nodes 12-20, light grey) and 6 mobile nodes (nodes 21-26, dark grey). The mobile nodes are initially placed near the sink in a sleep mode until needed.

Scenario 1. The initial topology of the scenario is presented in Fig. 7.4a. This topology was selected to demonstrate an example where the faults can disconnect the network, when crucial nodes fail. Fig. 7.4b shows the faults that happen in the network at nodes 4, 5, 6 and 7. When these nodes fail, the network disconnects and the detection mechanisms start running. After mobile node 21 is placed to the position of the first faulty node (node 4) and investigates the affected area, it will find the other faults as well. The sink will be informed by the mobile node about its findings and start the recovery phase.

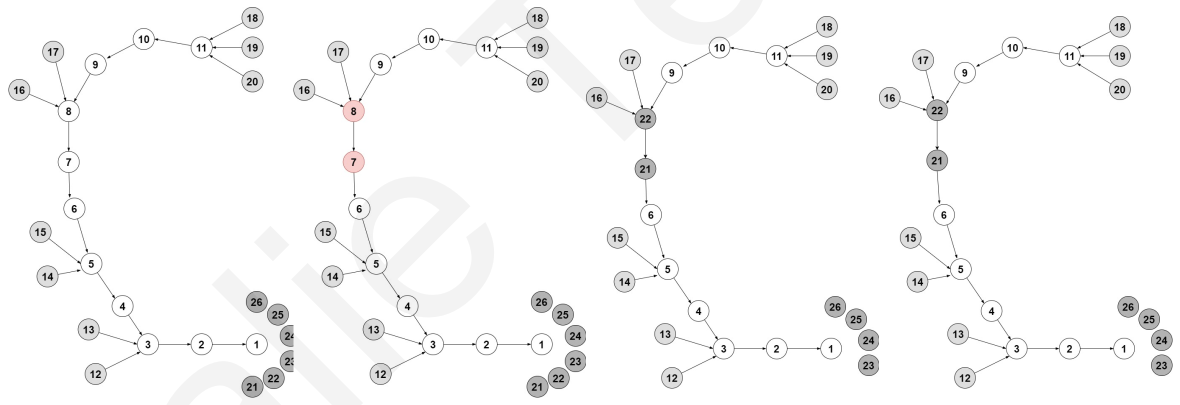
The recovery process presented in Fig. 7.4 shows the difference of the two algorithms. In the Replacement algorithm (Fig. 7.4c), three extra mobile nodes are used, as the number of the faulty nodes. In contrast, in the DFTNPA algorithm (Fig. 7.6d), only two extra mobile nodes are used, since the algorithm was able to find a position to cover both affected areas of the faulty nodes 6 and 7.



(a) Initial Topology *(b) Faults in Network* *(c) Replacement* *(d) DFTNPA*

Figure 7.4: Scenario 1

Scenario 2. The initial topology of this scenario is presented in Fig. 7.5a. This topology was selected to connect this algorithm with the previous ones, and have same results for the same topology. Fig. 7.5b shows the faults that happen in the network at nodes 7 and 8. When these two nodes fail, the network disconnects and the detection mechanisms start running. After mobile node 21 is placed to the position of the first faulty node (node 7) and investigates the affected area, it will find the second faulty node (node 8). The sink is informed by the mobile node of its findings and starts the recovery phase.

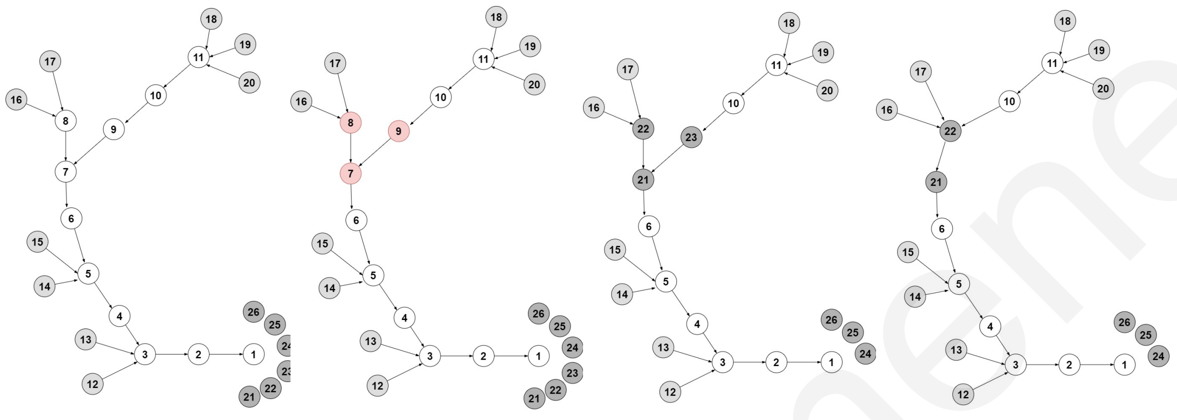


(a) Initial Topology *(b) Faults in Network* *(c) Replacement* *(d) DFTNPA*

Figure 7.5: Scenario 2

Both recovery methods (Fig. 7.5), Replacement and DFTNPA, use the same number of mobile nodes to recover the faulty state of the network. For resolving this scenario only one extra mobile node is needed. The difference of the two methods relays on the position of this mobile node. The Replacement algorithm (Fig. 7.5c) just locates the mobile node to the previous position of the faulty node, whereas the DFTNPA algorithm (Fig. 7.5d) finds a better position based on the distances of the affected nodes.

Scenario 3. The initial topology of the network is presented in Fig. 7.6a. This topology creates faults at critical nodes that disconnect both upper area nodes from the rest of the network. Fig. 7.6b shows the faults that happen in the network during the simulation at nodes 7, 8 and 9. Once the first fault is detected, the first mobile node is placed (mobile node 21) to investigate the area. Once all information are gathered to the sink node, the recovery phase starts.



(a) Initial Topology *(b) Faults in Network* *(c) Replacement* *(d) DFTNPA*

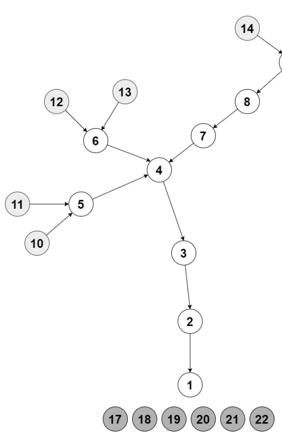
Figure 7.6: Scenario 3

The recovery process presented in Fig. 7.6 shows the difference of the two methods. In the Replacement algorithm (Fig. 7.6c), two extra mobile nodes are used, as the number of the faulty nodes. In contrast, in the DFTNPA algorithm (Fig. 7.6d), only one extra mobile node is needed, since the algorithm was able to find a position that can serve all affected nodes in the area.

CFTNPA Scenarios

We implemented six different scenarios with different number of faulty nodes and two different topologies. The first topology (see Fig. 7.7a) employs 22 nodes, where 1 sink node (node 1), 8 relay nodes (nodes 2-9, white), 7 source nodes (nodes 10-16, light grey) and 6 mobile nodes (nodes 17-22, dark grey). The second topology (see Fig. 7.7b) employs 31 nodes, where 1 sink node (node 1), 8 relay nodes (nodes 2-9, white), 7 source nodes (nodes 10-16, light grey) and 6 mobile nodes (nodes 17-22, dark grey). The first topology presents faults happening in the same area and the second topology presents faults happening in two opposite areas in the network The mobile nodes are initially placed near the sink in a sleep mode until needed.

Scenario 1. Fig. 7.8 presents the different states of Scenario 1. We present the solutions with the use of the Replacement algorithm and the CFTNPA algorithm, which is able to provide solution with two variations, towards an active node and the sink.


(a) Topology 1

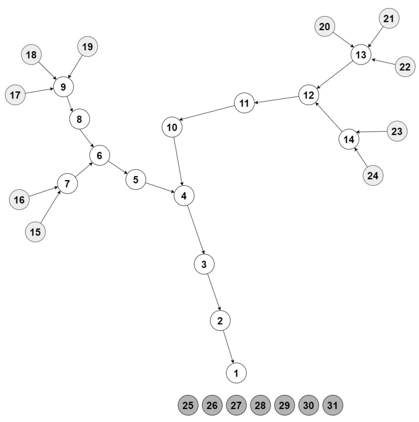
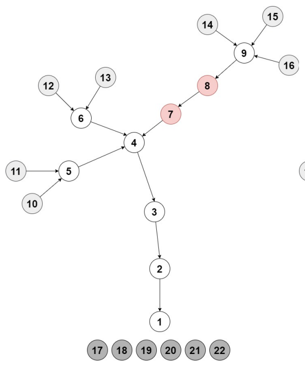
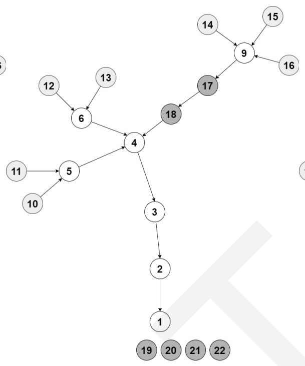
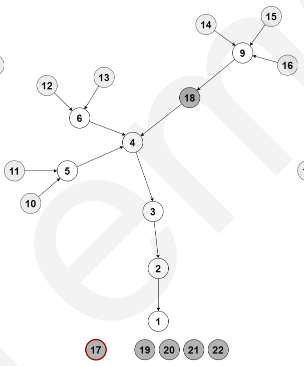

(b) Topology 2

Figure 7.7: Initial Topologies of Centralized Algorithm


(a) Faults in the Network


(b) Replacement


(c) CFTNPA - active node

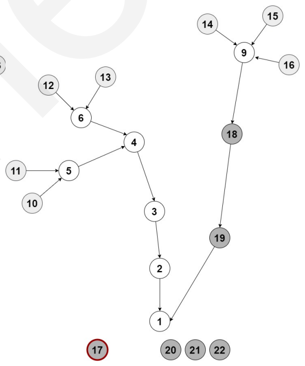

(d) CFTNPA - sink node

Figure 7.8: Scenario 1

Fig. 7.8a presents the faults in the network during the simulation at nodes 7 and 8. Once the sink realizes the absent of the source nodes affected from the faults, it will send a mobile node (mobile node 17) to investigate the area. When all information are gathered at the sink, with the return of the investigation mobile node, the recovery phase starts. In the Replacement algorithm (Fig. 7.8b), two mobile nodes are assigned to replaced the faulty nodes. However, in the CFTNPA Algorithm the problem can be solved with two different ways. The first solution shown in Fig. 7.8c will connect the affected area with an active node (node 4), where only one mobile node is needed to resolve the problem. In the second solution shown in Fig. 7.8d, the path will be created with destination the sink node, which will use two mobile nodes for reconnecting the affected nodes and provide them with a new alternative path to the sink node.

Scenario 2. Fig. 7.9 presents the different states of Scenario 2. We present the solutions with the use of the Replacement algorithm and the CFTNPA algorithm, which is able to provide solution with two variations, towards an active node and the sink.

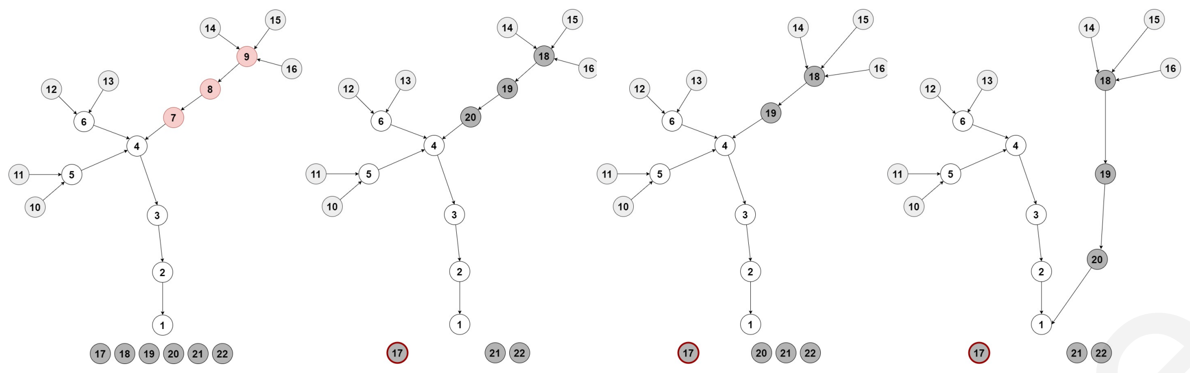

(a) Faults in the Network (b) Replacement Algorithm (c) CFTNPA - active node (d) CFTNPA - sink node

Figure 7.9: Scenario 2

Fig. 7.9a presents the faults in the network during the simulation at nodes 7, 8 and 9. Once the sink realizes the absent of the source nodes affected from the faults, it will send a mobile node (mobile node 17) to investigate the area. When all information are gathered at the sink, with the return of the investigation mobile node, the recovery phase starts. In the Replacement algorithm (Fig. 7.9b), two mobile nodes are assigned to replace the faulty nodes. However, in the CFTNPA Algorithm solution, we show two variations, a path towards an active node in the network and a path towards the sink node. (Fig. 7.9c) shows the path that uses two new mobile nodes to connect the affected nodes with an active node (node 4) in the network, whereas (Fig. 7.9d) shows the path that uses three new mobile nodes to connect the nodes direct to the sink node.

Scenario 3. Fig. 7.10 presents the different states of scenario 3. We present the solutions with the use of the Replacement algorithm and the CFTNPA algorithm for a two area fault in the same area of the network.

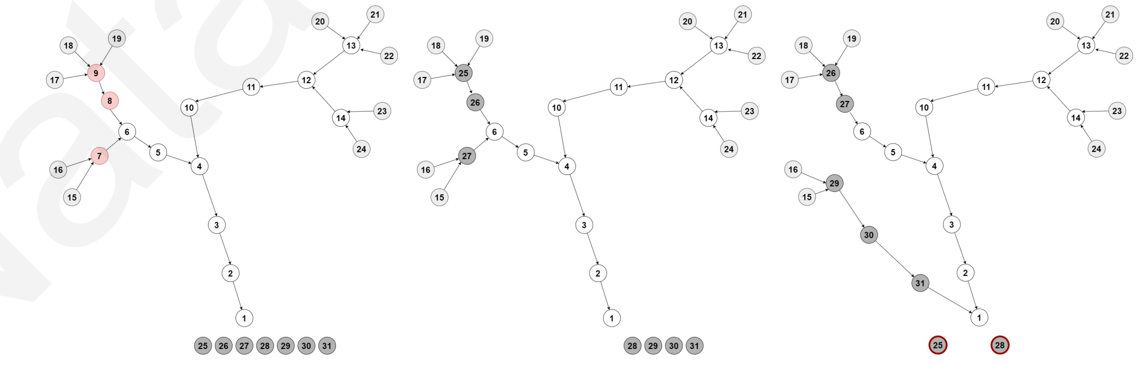
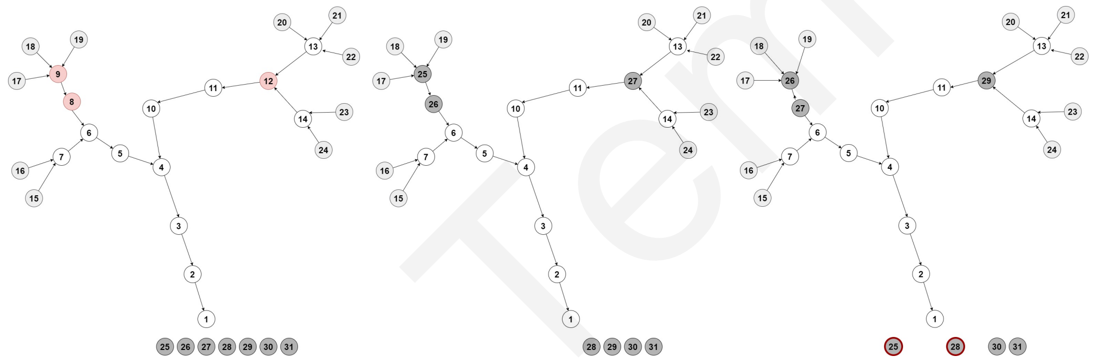

(a) Faults in the Network (b) Replacement (c) CFTNPA

Figure 7.10: Scenario 3

Fig. 7.10a presents the faults in the network during the simulation at nodes 7, 8 and 9, where first the nodes 7 and 8 will fail, and then node 9. Once the sink realizes the absent of the source nodes affected from the faults, it will send a mobile node (mobile node 25 and 28) to investigate the areas. When all information are gathered at the sink, with the return of the investigation mobile nodes, the recovery phase starts. In the Replacement algorithm (Fig. 7.10b), each faulty node is replaced with a mobile node, which means that in total three new mobile nodes are employed in the network. However, in the CFTNPA Algorithm (Fig. 7.10c), for the first faulty area, two mobile nodes are used in a calculated position that is optimal than the one of the faulty nodes, and for the second fault, a new alternative path is created towards the sink node, that uses three mobile nodes.

Scenario 4. Fig. 7.11 presents the different states of scenario 4. We present the solutions with the use of the Replacement algorithm and the CFTNPA algorithm for a two area fault in different areas of the network.



(a) Faults in the Network *(b) Replacement* *(c) CFTNPA*

Figure 7.11: Scenario 4

Fig. 7.11a presents the faults in the network during the simulation at nodes 7, 8 and 12. The first area of fault is due to the faults of nodes 7 and 8, and fault at node 12 affects the second area. Once the sink realizes the absent of the source nodes affected from the faults, it will send two mobile nodes (mobile node 25 and 28) to investigate the two areas. Each time one of the investigation mobile nodes returns to the sink with the information needed, the recovery phase starts. In the Replacement algorithm (Fig. 7.11b), each faulty node is replaced with a mobile node, which in total uses three mobile nodes. Although the solution provided with the CFTNPA Algorithm (Fig. 7.11c) also uses three mobile nodes, their positions are different than the previous nodes. The mobile nodes are placed in a more optimal position than before.

7.3.3 Evaluation Metrics

The DFTNPA and CFTNPA algorithms were evaluated using six metrics: the number of mobile nodes used, the total distance, the total time, the percentage of successfully received packets, the total packet loss, and the total energy consumption. Recall from Section 6, the algorithms are evaluated based on a single-valued measure of performance.

Total Number of Mobile Nodes. The total number of mobile nodes presents the number of all mobile nodes that are used throughout the simulation for solving the problems that occur in the network.

Total Distance. The total distance presents the distance of each mobile nodes that has traveled in the network for resolving a problem that has occur. The distance for each route a mobile node does is calculated with the euclidean distance from each starting point to its destination point and the total amount is calculated based on the sum of each used mobile node in the network with the equations below:

$$Total\_Distance = \sum_{i=1}^{M} dist_{m_i}$$

where, $dist_{m_i}$ is the distance of mobile node m_i and is calculated based on the euclidean distance, with the equation: $dist_m = \sqrt{((x_{dest} - x_{start})^2 + (y_{dest} - y_{start})^2)}$. The point (x_{dest}, y_{dest}) represent the destination position of the mobile node and point (x_{start}, y_{start}) represent the starting position of the mobile node.

Total Time. The total time presents the total time the network needs to reconnect and return to a stable state, and is presented with the equation below:

$$Total\_Time = t_2 - t_1$$

where, $t1$ represents the time where a fault is detected in the network and enters a faulty state, and $t2$ represents the time where the network reconnects and the packets from the affected area(s) are received by the sink node.

Percentage of Successfully Received Packets. As mentioned in Chapter 6, the percentage of successfully received packets presents the ratio of the successfully received packets versus the total sent packets and is calculated with the equation below:

$$Recv\_Pkts\_Ratio(\%) = \frac{successfully\_received\_packets}{total\_sent\_packets}$$

Total Packet Loss. The total packet loss presents the ratio of the difference from the total number of transmitted packets (T_x) and the total number of received packets (T_r), with the total number of transmitted packets (T_x), and is calculated with the equation below:

$$Packet\_Loss\_Ratio = \frac{T_x - T_r}{T_x}$$

Total Energy Consumption. As mentioned in Chapter 6, the total energy consumption is measured in mJ and is defined as the sum of the computational energy ($energy_{i_{listen}}$) and the moving energy ($energy_{i_{move}}$) of each node. Only the mobile nodes have a moving energy, for the other nodes the moving energy is set to zero. The equation is presented below:

$$E_{total} = \sum_{i=1}^{N} energy_{i_{listen}} + energy_{i_{movie}} \tag{7.1}$$

The $energy_{listen}$ represents the energy computational usage and is calculated with the equation from [66]:

$$\begin{aligned} listening\_energy_i = (&transmit \cdot 19.5mA + listen \cdot 21.8mA + \\ &CPU \cdot 1.8mA + LPM \cdot 0.0545mA) \cdot 3V/4096 \cdot 8 \end{aligned}$$

where *trasmit* is the total time of the radio transmitting, *listen* is the total time of the radio listening, *CPU* is the total time of the CPU being active, and *LPM* is the total time of the CPU being in low power mode.

The $energy_{move}$ represents the energy usage of moving, for all node types the moving energy is set to zero expect the carrier and is calculated with the equation from [96]:

$$energy_{i_{move}} = P_u \cdot \frac{s}{u} \tag{7.2}$$

where P_u is the power consumption of a given speed u and s is the total traveling distance.

Computational Time. As mentioned in Chapter 6, the computational time refers to the time needed for the sink node to calculate the position of the mobile node. For the reasons explained in Section 6.2, we do not include it in the results.

7.3.4 Evaluation Results

In this section, we present the experimental results of our evaluation. We start by presenting the comparison of the Decentralized Node Placement Algorithm with the Replacement algorithm. Then we present the results of the Centralized Node Placement Algorithm compared to the Replacement algorithm.

DFTNPA Evaluation

For the execution example in Section 7.3.2, we also present some basic experimental results.

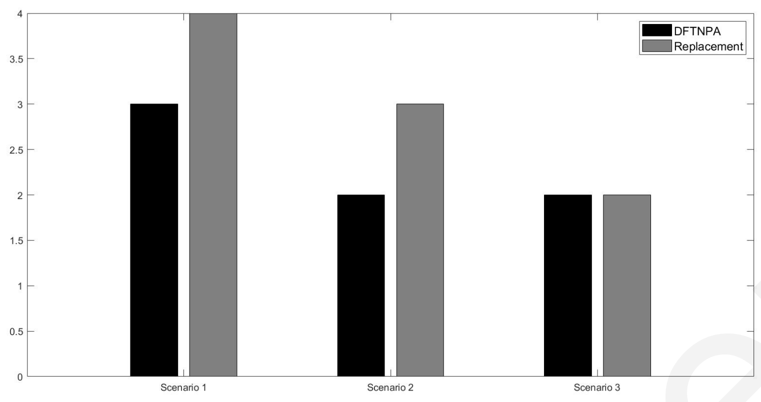


Figure 7.12: Number of Mobile Nodes used

Fig. 7.12 shows the number of mobile nodes used for each scenario. This plot shows that the Replacement algorithm uses more mobile nodes than the DFTNPA algorithm. The results are expected as in the DFTNPA algorithm each position calculated is examined to be located in such a way to cover as many affected nodes that are in range as possible. The Replacement algorithm just replaces the faulty nodes with mobile nodes without considering the possibility that a new position can be used to assist more nodes.

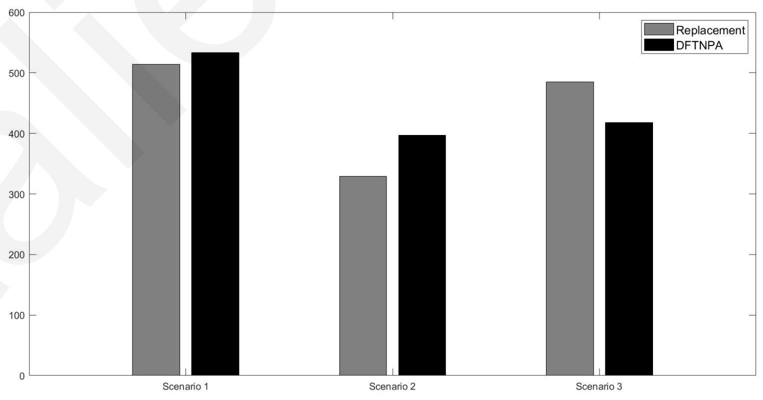


Figure 7.13: Total Distance

Fig. 7.13 shows the total distance the mobile nodes traveled in the network during the whole simulation. This plot shows that the mobile nodes in the Replacement algorithm travel a smaller distance than the DFTNPA algorithm. This is normal as in the latter a discovery process is performed by the first placed mobile node in the affected area as requested by the sink, where the mobile node travels to suspected non-alive nodes to investigate their status.

Whereas in the former, the mobile nodes only travel to the positions of the faulty node after assigned from the sink node. As a result, in all scenarios, the DFTNPA algorithm results in higher values than the Replacement algorithm, even though a smaller number of mobile nodes are used, as in Scenarios 1 and 3, or better positions are calculated, as in Scenario 2.

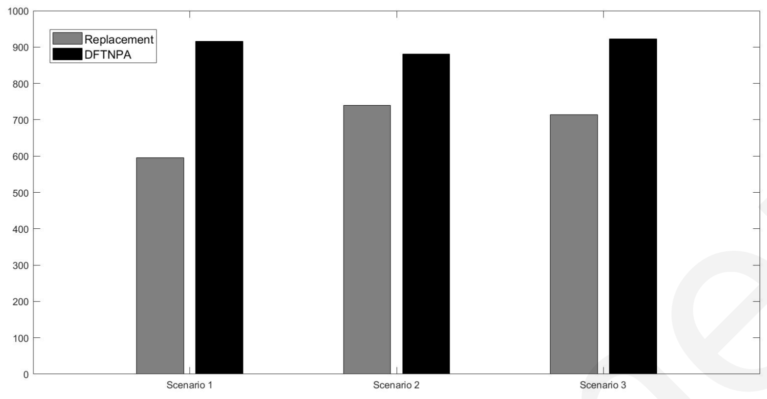


Figure 7.14: Total Time

Fig. 7.14 shows the total time the network needs to reconnect and return to a stable state. This plot shows that in the Replacement algorithm the network returns to a stable state quicker than the DFTNPA algorithm. The Replacement algorithm only finds the position of the faulty nodes and assigns them to the new mobile nodes. However, in the DFTNPA method, the sink node needs to calculate the new positions of the mobile nodes based on parameters which require more time, and then assign these positions to the mobile nodes. As a result, the extra amount of time needed from the DFTNPA method is the calculation procedure of the sink node for the new positions of the mobile node. It can be also observed that the time in the DFTNPA is also on from the number of faulty nodes, which means that more faulty nodes results in more affected nodes that need to be processed to find a position for a mobile node.

Fig. 7.15 presents the percentage of successfully received packets by the sink node during the simulation. This plot shows that the Replacement algorithm resolves the problem occurred in the network faster than the DFTNPA algorithm, since its procedure is simpler, where the new node is placed on the current position of the faulty node. All scenarios start by acting the same and fail with the same numbers, but differentiate when the recovery process starts to run. Scenario 1 is the one that reaches the lowest throughput during simulation for both algorithms. This happens because in this scenario the affected area of the faulty nodes is in such a position that almost every node loses its connection to the sink. This is also the reason why it needs more time to return to a stable state. It is observed that the parameter

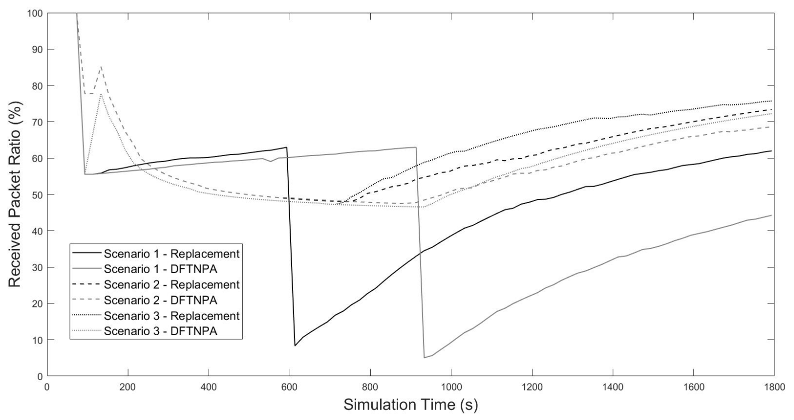


Figure 7.15: Percentage of Successfully Received Packets

that affects this metric is not only the number of faulty nodes, but also the position of the faults. When a bigger part of the network is affected from the faults, more time is needed to return to a stable state. In both scenarios, 2 and 3, the DFTNPA algorithm needs more time to increase the number of received packets, because it needs more time to calculate the position of the mobile nodes.

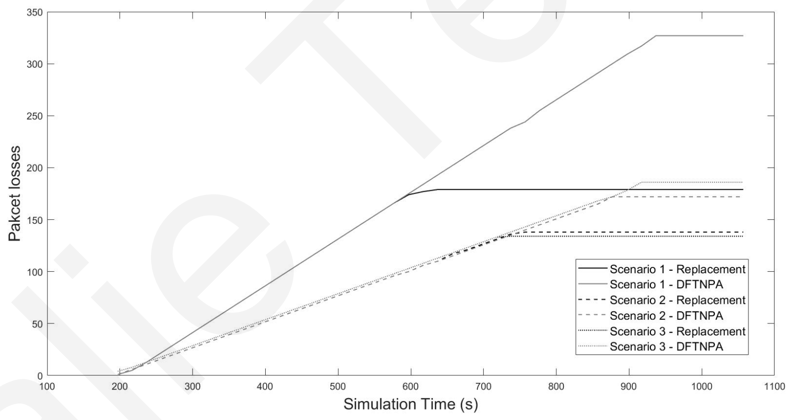


Figure 7.16: Total Packet Loss

Fig. 7.16 presents the total packet loss during the simulation. This plot shows that, as mentioned before, the Replacement algorithm reconnects the network faster so the stabilization of the lost packets in the network happens quicker than the DFTNPA algorithm. It is noticeable that the number of lost packets is again defined based on the position of the fault and the number of affected nodes. Scenario 1 presents the most lost packets because it has the highest number of faulty nodes and their positions are critical. Whereas, Scenario 3 has the smallest number of lost packets as it has the smallest number of faults.

Fig. 7.17 shows the total energy consumption for each scenario. It is observed that in all scenarios of the DFTNPA algorithm fewer energy is consumed in comparison with the

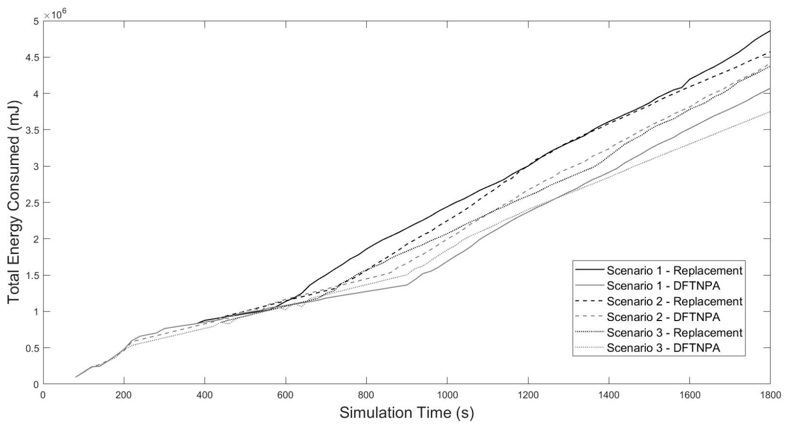
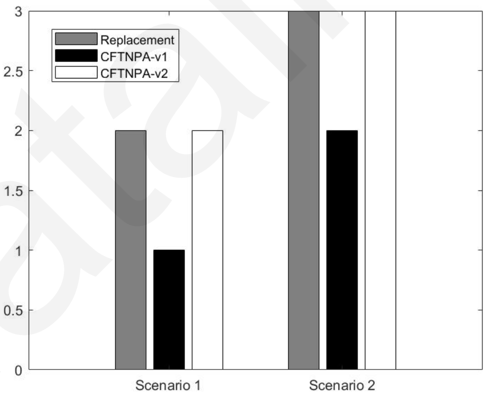


Figure 7.17: Total Energy Consumed

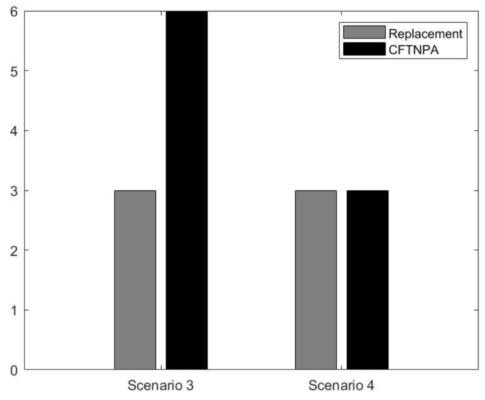
Replacement algorithm. This happens because the number of mobile nodes used is lower. Another reason is that the DFTNPA algorithm calculates a new position, which is in a more optimal location. As a result, the distance from the original position is not the same, and in most occasions the original is further than the new one. The most consumed energy is in the Replacement algorithm of Scenario 1, which was expected since it is the scenario with the most faults; thus more mobile nodes are used. The best results are presented by Scenario 3 for the DFTNPA algorithm, where few faults occurred and the recovery process uses less mobile nodes, as well as their positions are closer to the sink.

CFTNPA Evaluation

For the execution example in Section 7.3.2, we also present some basic experimental results.



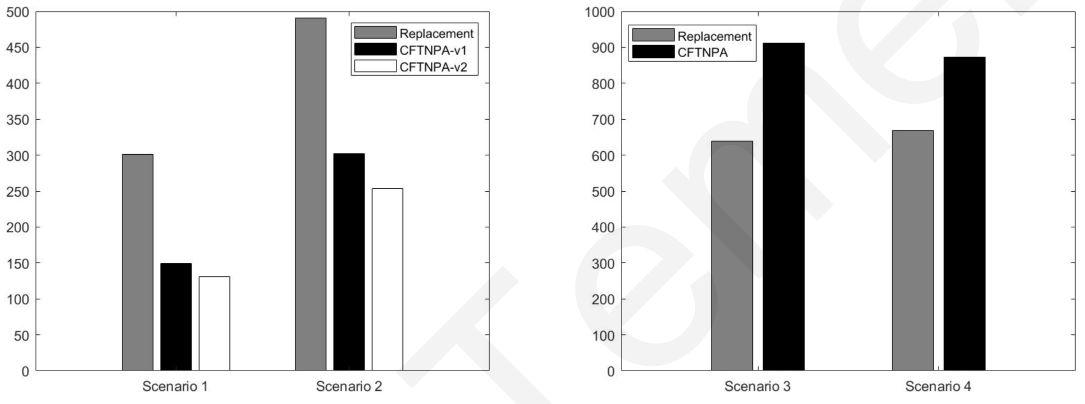
(a) One area of Failure



(b) Two areas of Failure

Figure 7.18: Number of Mobile Nodes used

Fig. 7.18 shows the number of mobile nodes used for each scenario. Fig. 7.18a presents the results for the one area failure scenarios (scenario 1 and 2). This plot shows that the solution of creating a new path towards an active node needs less mobile nodes than the others. The Replacement algorithm uses as many mobile nodes as the number of faulty nodes occur in the network. The solution with a path towards the sink uses more mobile nodes than the other solution and same as the replacement, which is not always the case. Fig. 7.18b presents the results for the two area failure scenarios (scenario 3 and 4). The plot shows that in the CFTNPA algorithm the number of mobile nodes it uses depends on the distance of the affected area to the sink node. Smaller distances will need smaller number of mobile nodes, whereas bigger distances need more mobile nodes, even more than the Replacement algorithm, which uses the same number as the faulty nodes.

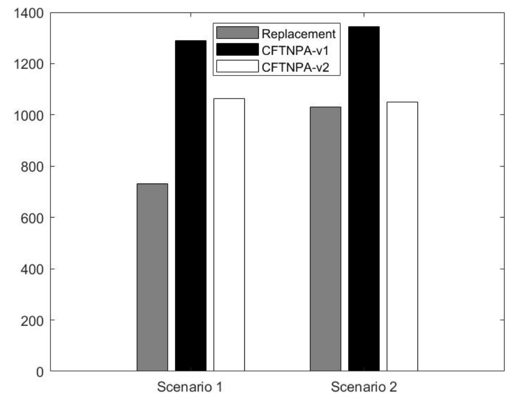


(a) One area of Failure *(b) Two areas of Failure*

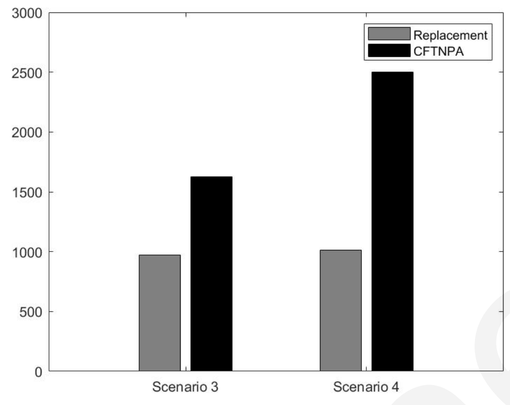
Figure 7.19: Total Distance

Fig. 7.19 shows the total distance the mobile nodes travel for resolving the problem in the network. Fig. 7.19a presents the results for the one area failure scenarios (scenario 1 and 2). This plot shows that the Replacement algorithm travels the most distance than all the solutions, as the positions of the mobile nodes are the same with the faulty ones. Because the CFTNPA algorithm calculates more optimal positions, both versions provide smaller results. The sink path solution provides the smallest distance since the mobile nodes closer to the sink will travel a smaller distance. Whereas, in the active node path creation, the distances are around the affected area. Fig. 7.19b presents the results for the two area failure scenarios (scenario 3 and 4). The plot shows the CFTNPA algorithm provides bigger results than the Replacement algorithm, which is normal because the CFTNPA uses more mobile nodes.

Fig. 7.20 shows the total time the network needs to reconnect the affected area with the sink node. Fig. 7.20a presents the results for the one area failure scenarios (scenario 1 and 2). This plot shows that the Replacement algorithm needs less time than the other solutions, as the algorithm just replace the faulty nodes. The active path variation of CFTNPA algorithm



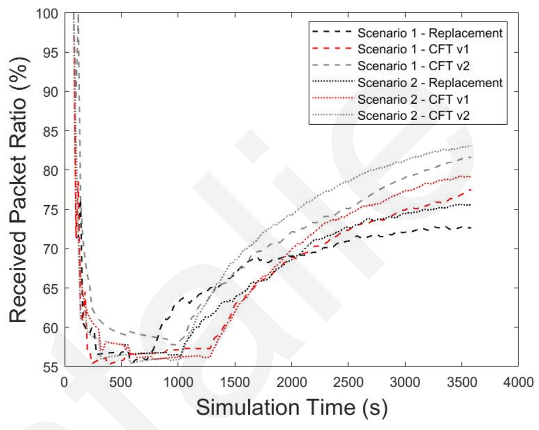
(a) One area of Failure



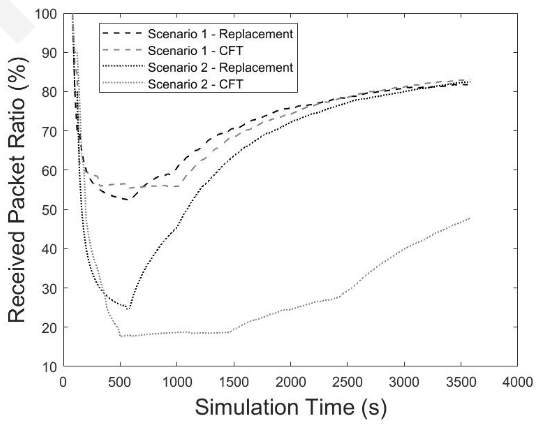
(b) Two areas of Failure

Figure 7.20: Total Time

needs more time than the other variation, because the calculations of the new positions need more time. Whereas in the alternative path towards the sink, only the first mobile node placed needs more complex calculation to be placed. Fig. 7.20b presents the results for the two area failure scenarios (scenario 3 and 4). The plot shows that the Replacement algorithm needs less time to resolve the problems than the CFTNPA algorithm. This is normal as the CFTNPA algorithm needs to calculate the positions the mobile nodes will be placed.



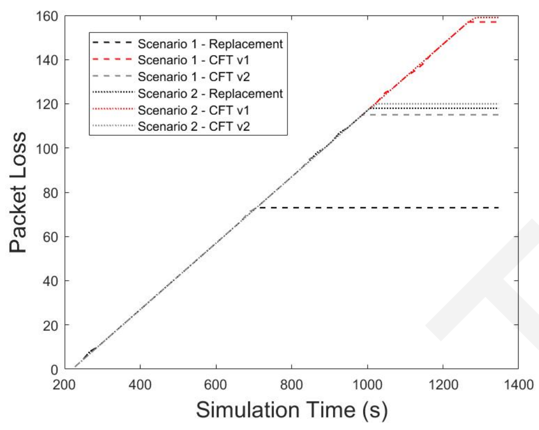
(a) One area of Failure



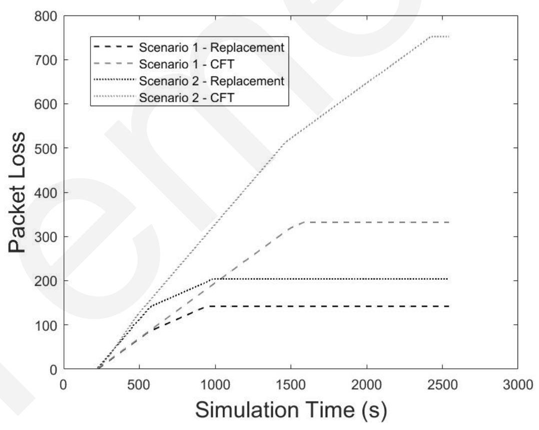
(b) Two areas of Failure

Figure 7.21: Percentage of Successfully Received Packets

Fig. 7.21 shows the percentage of successfully received packets by the sink node during the simulation. The one area plot (Fig. 7.21a) shows that the CFTNPA algorithm that creates a direct alternative path towards the sink provides the most received packets after recover than the other scenarios. This happens because the new path will not interfere with the existing network nodes. The Replacement algorithm even though the recovery is happening sooner than the others, due to interaction with the existing nodes and collisions that happen in the network provides the worst results after the recovery. The scenario with the local solution, path towards an active node, even though it recovers faster than the alternative path, it needs more time to restore the received packets. In the two area plot (Fig. 7.21b), we can observe that the faults in scenario 3 have little affect than scenario 4, due to the position of the failures, less packets are missed. In both scenarios, it is shown that the Replacement algorithm recovers the network quicker than the CFTNPA algorithm, as the calculations of the positions of the mobile nodes needs more time. Except that, we can observe that Scenario 4 needs more time to recover than Scenario 3, because in the former the two failures are in two different areas of the network and in the latter they are happening in a smaller distance between them.



(a) One area of Failure

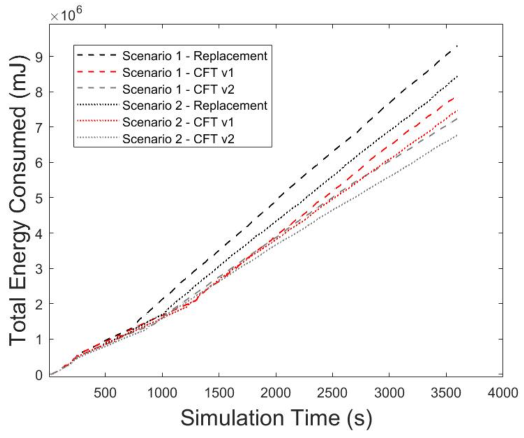


(b) Two areas of Failure

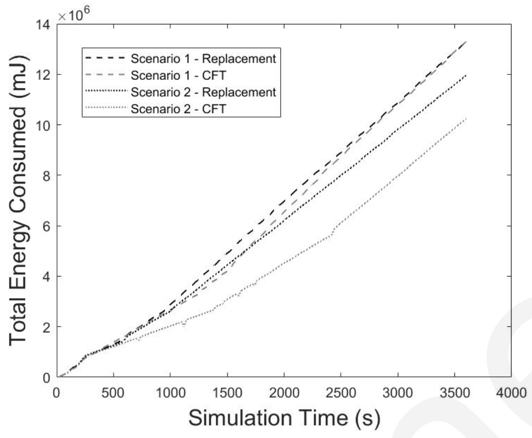
Figure 7.22: Total Packet Loss

Fig. 7.22 shows the total packet loss during the simulation. The one area plot (Fig. 7.22a) shows that the quickest recovery is made by Scenario 1, and more specific by the Replacement algorithm. The smaller time needed by some of the scenarios is also based on the number of faults that happen in the network. The Scenario with the smallest faults needs fewer time to recover, that is the reason why all versions of Scenario 1 are better than the ones from Scenario 2. The versions that create alternative paths recover faster than the one with the local solution, as the calculation of the path is easier. In the two area plot (Fig. 7.22b), we can observe that again the Replacement algorithm recovers faster than the CFTNPA algorithm. Scenario 4 needs more time to recover as the faults occurring in the network are in two different parts of it and need more time to be investigated and recover, whereas the investigation process in Scenario 3 is easier as the faults are in the same part of the network.

Fig. 7.23 shows the total energy consumption of the nodes in the network during the simulation. The one area plot (Fig. 7.23a) shows that even though the Replacement algorithm



(a) One area of Failure



(b) Two areas of Failure

Figure 7.23: Total Energy Consumption

uses the same nodes as before the faults, it consumes the most energy, and this happens because its recovery process is faster than the CFTNPA algorithm so the packet transmission restarts faster. The CFTNPA algorithm version of the alternative path solution has smaller consumption than the local solution of the algorithm, as the new path relieves the area from the extra resources that now use this path. In the two area plot (Fig. 7.23b), we can observe the same results as in the previous plot. The CFTNPA algorithm shows smaller consumption than the Replacement algorithm, because it needs more time to recover from the faults in the network, so the normal transmission process starts later on in the simulation.

7.3.5 Comparison between DFTNPA and CFTNPA

Based on the evaluation results for both algorithms (DFTNPA and CFTNPA), it is observed that the use of mobile nodes can efficiently alleviate a fault that occurs in the network. As each algorithm uses a different detection and recovery mechanism, it can be noticed that both algorithms have their advantages and disadvantages, which are based on different scenarios. Algorithm DFTNPA is able to decrease the amount of extra resources used and still recover the network, whereas Algorithm CFTNPA with the creation of an alternative path, especially, towards the sink node, is able to relieve the affected area from extra load. However, the calculations for the position of the mobile nodes is using more time, which increases the time of the network's recovery process. It needs to be clarified that Algorithm Direct MobileFT, which is used for creating alternative paths of mobile nodes, initially uses Algorithm Dynamic MobileFT for calculating the position of the first mobile node of the path. Even though the procedure of Algorithm Direct MobileFT is simple after the first step, it needs a little more time for its calculation than Algorithm Dynamic MobileFT. This means that in general, Algorithm DFTNPA is able to recover faster than Algorithm CFTNPA.

Both algorithms have been compared with the Replacement Algorithm and shown that even though the Replacement Algorithm is faster in resolving a problem, as it just replaces each faulty node, Algorithms DFTNPA and CFTNPA are able to provide a better solution for a long term period. Algorithm DFTNPA may in some cases use the same or even fewer amount of mobile nodes as the Replacement Algorithm, but the position of the mobile nodes placed in the network is calculated in such a way by the sink node that it is able to serve as many nodes as possible. This means that the new position is better in terms of location and ability to serve the affected nodes, than the one of the faulty node. Whereas, Algorithm CFTNPA is able to extend this solution by providing a new path by using only mobile nodes with destination either the sink node or an active node in the network. This approach is able to unload the affected area by adding a new option to the nodes, a new path towards the sink node.

Algorithm DFTNPA is used for resolving a fault in a local way, whereas Algorithm CFTNPA resolves the fault in a more global way. In terms of detection, the detection mechanism of Algorithm CFTNPA is faster than the one of Algorithm DFTNPA. This happens because in DFTNPA the neighboring node is responsible of detection, which means that the nodes, except that they need time to identify the fault in their neighborhood, they also need time to inform the sink node about its findings. The detection method can be divided into two steps. The first step is responsible to detect the fault in the network and the second step to inform the sink node about this fault. Algorithm DFTNPA needs to follow both steps, whereas Algorithm CFTNPA only needs to detect the fault, as the detection is done from the sink itself. However, in terms of the investigation process, where the affected area and nodes are found, Algorithm DFTNPA has more accurate information than Algorithm CFTNPA. The neighboring node of a faulty node is able to provide more detailed information to the sink about the fault, rather than the general view of the sink node. This view is more limited in scenarios where the static nodes are divided into source and relay nodes (which is the assumption of our scenarios). In these cases, the sink node is only able to detect the source nodes that it can not receive, however, these nodes may not be faulty, but the fault may be in a different part of the path from these source nodes. Thus, the investigation process to identify the fault is more time consuming.

In terms of recovery, Algorithm DFTNPA uses a local method to recover from the fault, whereas Algorithm CFTNPA creates a new path of mobile nodes towards the destination node. Algorithm DFTNPA only calculates a new position for the mobile node that is going to be placed in the affected area based on the needs of the affected area. Algorithm CFTNPA starts with the same step as Algorithm DFTNPA, where it uses the same algorithm for placing the first mobile node, but then it creates a path from this mobile node towards the destination node by placing more mobile nodes. The destination node can be either directly the sink node or a lower level active node in the network. In both cases, the path is able to unload the affected area and create a direct path towards the sink node. The creation of an alternative path of mobile nodes means that more resources are employed in the network, which results in consuming more energy and needing more time for resolving the problem. Although, these parameters are a disadvantage for Algorithm CFTNPA, at the same time they can be an advantage in cases where the failure is able to affect the whole neighborhood and it is needed to bypass this part of the network. This is the time a new path is needed to work the way towards the sink node. This approach is also able to reduce packet loss because the new path is able to communicate either directly or with fewer hops to the sink node.

Chapter 8

Conclusions

8.1 Summary

In this doctoral dissertation we have presented algorithms that utilize mobile nodes in assisting the network with the occurrence of a problem. The mobile nodes are sent to the affected area to assist either the affected or the affecting nodes. Additionally, we introduced the term of reuse, by reusing the mobile nodes already placed in the network if possible.

Initially, we presented the Node Placement Algorithm (NPA) with two variations, which gets initiated when existing congestion control algorithms fail. The algorithm employs mobile nodes to either create disjoint paths of mobile nodes and route the excess traffic directly to the sink (Direct MobileCC), or to place a mobile node in such a position to create an alternative path by bridging two disjointed areas in the network and repeating the process if necessary (Dynamic MobileCC). Simulation results demonstrate that both variations can alleviate congestion. In doing so, Direct MobileCC demonstrates a better average source to sink delay and reduced packet drop, at the expense of mobile nodes used (almost double) and energy consumed, when compared to Dynamic MobileCC. In this part of the work, we have considered one instance of using alternative paths for alleviating congestion.

Next, we extended the previous concept by considering the energy consumption of the mobile nodes used in the network to alleviate congestion in WSNs. We introduced the term of reuse by extending the Dynamic MobileCC algorithm into an energy-efficient solution, called Energy-aware Node Placement Algorithm. The extended version considers the energy consumption of the nodes in the network, where the mobile nodes added to the network use their energy level for every decision that needs to be made. This action is helpful in replacing an energy exhausted mobile node in time before new congestion occurs or in reusing it to a new position for resolving new congestion. The simulation results demonstrate that the proposed algorithm can effectively alleviate congestion in the network. The two methods used for allocating an alternative node show slight differences in the results, which is expected due to their algorithmic structure, where the allocation method uses more information messages and computation than the optimal method. In the case of the scenarios, it is noticeable that the least effective one is the one where no reuse occurs, with its result being the lowest in terms of packets and the highest in terms of energy consumption. The other scenarios have similar reactions in terms of packets, whereas based on the energy consumption, they all have different reactions based on the number of mobile nodes that are used and needed. We also evaluated this algorithm under different energy models. Based on the results, it is demonstrated, as expected, that the energy consumption of the network increases based on the number of nodes. When more nodes are active in the network, more energy is consumed. Additionally, when the energy model uses the time variable, it is noticeable that the results are increasing, as this variable changes constantly, whereas the distance variable changes only when a node moves, which is rarer. In general, it is demonstrated that our solution is able to be used effectively with different mobile robots.

Then, we extended the previous concept by considering the use of mobile carriers that can transport mobile nodes around the network and position them to the requested location and present the Carrier-based Node Placement Algorithm. The case study of resolving congestion control problems in the network was presented as part of the evaluation of the algorithm. Our experimental evaluation shows that the two approaches (stay or leave), depending on the specific scenario, one can outperform the other. The Stay Approach, although consuming more energy, it can be a good solution for cases where the return of the carrier is hard. On the other hand, the Leave Approach can be implemented in environments where the carrier needs to return to its initial position, either because it is too dangerous to stay or few carriers are available. Both factors were responsible for the results of each scenario. In terms of delay and throughput, the reuse scenario outperformed the others because the network is re-established in a shorter time. Regarding the energy consumption, the replace scenario outperformed the other because fewer extra resources were needed. Our findings have shown that the proposed algorithm uses the energy consumption of the mobile node to the fullest without considering the returning process, which is a part of the carrier, while the carrier performed their transportation effectively.

Finally, we extended the problem of interest and used mobile nodes for resolving the occurrence of faults in the network. We presented the Fault-Tolerant Node Placement Algorithm (FTNPA) that consists of two variations, the Decentralized Fault-Tolerant Node Placement Algorithm (DFTNPA) and the Centralized Fault-Tolerant Node Placement Algorithm (CFTNPA). The DFTNPA algorithm uses a decentralized detection mechanism, where a node in the network is responsible in identifying a faulty node in its neighborhood, and a local recovery mechanism, where a mobile node is placed in such a position to serve the affected nodes using the Dynamic MobileFT algorithm. The CFTNPA algorithm uses a centralized detection mechanism, where the sink node is responsible to identify a faulty node in the network from the nodes that are sending packets to it, and a recovery mechanism that creates alternative paths towards either the sink node or an active node of the network, with the use of the Direct MobileFT algorithm. The evaluation of these algorithms shows that both algorithms have their advantages and disadvantages based on the scenario. The DFTNPA algorithm is able to decrease the amount of extra resources used and still recover the network, whereas the CFTNPA algorithm, with the creation of an alternative path, especially, towards the sink node, it is able to relieve the affected area from extra load. However, the calculations for the position of the mobile nodes is using more time, which increases the time of the network's recovery process. We need to clarify that the Direct MobileFT algorithm used for creating alternative paths of mobile nodes, firstly uses the Dynamic MobileFT for the positioning of the first mobile node of the path. Even though the procedure of the Direct one is then simple, it needs a little more time for its calculation than the Dynamic one. In long term, it means that the DFTNPA algorithm recovers faster than the CFTNPA algorithm.

8.2 Future Work

The work presented in the thesis has several interesting future directions. Some are shorter-term (Section 8.2.1), while other are longer-term (Section 8.2.2).

8.2.1 Short-term Extensions of the Thesis Work

The thesis work could be further extended in different directions in the near future. These directions are presented below:

- *Scalability*: Since the algorithms are local and dynamic, one would expect that they scale well with network size, number of mobile nodes, and number of failures. Future experimental evaluation could validate this expectation. As the COOJA simulator cannot support large number of nodes (due to resource limitations), such experimentation could occur on MATLAB or NS3. Thus, the algorithms we will first need to be implemented on these platforms.

- *Hybrid detection mechanism*: In the MobileFT Framework, the detection method of a faulty node can be performed either by a neighboring node or the sink node. An extension of this work is currently directed towards a hybrid mechanism of detection that combines both a neighboring node and the sink node for faster detection. These two mechanisms can be used for different cases of faults based on the advantages each mechanism can provide. For the recovery mechanism, mobile nodes are used either as a local solution or by creating an alternative path towards a destination, based on the detection mechanism used.

- *Energy-aware FTNPA solution*: The existing FTNPA algorithms consider energy only as a failure (when the nodes exhausts its energy). However, it would be interesting to consider the energy consumption in the same level of granularity as done with the eNPA algorithm, and investigate how it would affect the algorithms' performance.

- *Different clustering algorithms*: As mentioned, for the purposes of the FTNPA algorithms, we have considered the k-means clustering algorithm. It would be interesting to consider other clustering algorithms and investigate how the use of a clustering algorithm affects the performance of FTNPA.

8.2.2 Long-term Extensions of the Thesis Work

In this section, long-term research directions are presented that are more challenging that the ones discussed in the previous section.

Handling Malicious Nodes and Attacks

An important and challenging issue for WSNs and IoT networks is security. The development of strong security algorithms is restricted due to resources limitation as well as the deployment environment. The network security can be threatened from data modification and malicious attacks that can be performed during the communication between the source nodes and the destination node. As a result, security is an essential component in dynamic networks that needs to be further researched.

An overarching goal on this work would be to explore in a multidisciplinary way the prominent areas of Security, Mobility and the Internet of Things. The main goal will be to implement self-recovery and network recovery with the use of mobile elements based on a range of different attacks. A possible solution is to use existing work that detect the abnormalities in the network and use the mobile nodes in a suitable position to avoid this situation. In [23–25] a monitoring tool and a detection mechanism are proposed. The first work [24] presents the Run-time Monitoring Tool (RMT) and its purpose is to monitor the operation of the nodes in the network based on specific parameters. The second work [25] presents a general methodology of an anomaly-based Intrusion Detection System (IDS), called mIDS and its purpose is to use the Binary Logistic Regression (BLR) as a statistic tool in order to classify the nodes of the network into benign and malicious. The main idea is to use RMT as the monitoring tool to get information about the nodes behavior that will be the input of mIDS. Then mIDS will derive a normal behavior model in order to detect abnormalities of the nodes. The third work [23], present an evaluation of the feasibility of running a lightweight Intrusion Detection System within a constrained sensor or IoT node. In this paper, mIDS is tested in a setting that contains network-layer attacks. The main idea is to use this system, where critical data are obtained from the routing layer and are then used as a basis for profiling sensor behavior. The results show that although the implementation is lightweight, the attack detection accuracy levels achieved a range of 96% - 100%. Using these algorithms in combination with the mobile nodes, it will be interesting to investigate whether it is possible, by placing the mobiles nodes in an appropriate way, to avoid abnormalities of the network. Additionally, we will investigate the attack groups our approach can be applied to.

Wireless Chargers

In recent years, research in WSN has been extended based on the movement capabilities of nodes in the network. Mobility has been introduced either at the sink or the nodes. Mobile sink approaches can be used to avoid congestion and at the same time increase the lifetime of the network. This approach could be useful for many applications, like target tracking, habitat monitoring and emergency preparedness. On the other hand, mobile nodes are useful for the network to stay connected. They assist in congestion mitigation and node replacement. When congestion is detected, mobile nodes can be placed in the network to take the excess traffic from the congested node. The lifetime of a network is limited based on the use of battery, but with the use of mobile nodes it can be maximized. This can happen either by replacing the low energy nodes with mobile nodes or by using mobile nodes to transport new nodes at the location.

As discussed above, it is noticeable that the lifetime of a network is limited due to the use of battery-based sensor nodes. A node that reaches to its lower energy level cannot recover, and as a result the network at some point will disconnect. In the literature, a recent direction of research consider *chargers* [64]. A charger is a high energy supply device that acts like a transmitter and it is responsible for the energy management in the network. This approach can be divided into two main categories, based on the number of chargers in the network, *single* or *multiple* chargers, and based on the ability of the chargers to move or not, i.e., *static charger* or *mobile charger*. A static charger is located at a specific point in the network and it will be helpful for mobile nodes. On the other hand, a mobile charger will move around the network and therefor be helpful not only for mobile nodes but also for static nodes. In [45], [46] the use of mobile chargers is considered. The network consists of static sensor nodes, a sink node and mobile chargers. As mentioned, a mobile charger is an entity with the ability to move around the network and wirelessly replenish the energy of the sensor nodes in the network. The work in [45] proposes four protocols where the two perform distributed, limited network knowledge coordination and charging, and the other two perform centralized, global network knowledge coordination and charging. The work in [46] proposes a hierarchical charging structure, by dividing the chargers into two groups, the *hierarchically lower mobile chargers* that charge sensor nodes and the *hierarchically higher special chargers* that charge mobile chargers. The works in [47], [48] provide solutions with the use of static chargers. The network consists of static nodes, a sink node, mobile nodes and static chargers. A static charger is an energy transmitter that will recharge a node when it gets in its range. Both works focus on dynamically selecting the appropriate charging range of a single static charger in the network. The main idea is that based on the energy of the charger, the corresponding energy can be transmitted to all of the nodes in its range, with respect to the node's battery life. The work in [57] proposes a way to exchange energy in a peer-to-peer way, without using special charger nodes. The mobile nodes used are equipped with a battery cell, a wireless transmitter and a wireless energy receiver. The main idea is that whenever two mobile nodes meet by intersecting their paths, they are able to exchange energy between their battery cells with the use of an interaction protocol. Two protocols are presented, a simple protocol for weighted energy balance with a loss-less transfer and a more natural protocol with an energy transfer with loss.

An interesting open question, that arises from the above works is: “how could (mobile) chargers be utilized for mobile congestion control?". We aim to redesign our algorithms using mobile nodes in order to use chargers in an effective way. The charger approach can be done either by using a *static charger* or using *mobile chargers*. A static charger is a static node that is placed in the network that has also the ability of charging nodes near it. The challenge of this approach is an efficient placement plan of the static nodes in the network. The placement of the static nodes is important and must be done in a way that in case of congestion the mobile nodes needed in the network will be placed near the static chargers. Additionally, it would be interesting to investigate the initial placement of the mobile nodes to be near the static chargers instead of all of them residing at the sink node. This can be either applied to all the mobile nodes or to a part of them, where the rest of them could be placed as defined by the MobileCC framework, near the sink node. On the other hand, the use of mobile charger(s) in the network can be more helpful, in terms of not wasting more resources when the local charging method can be used. This means that when a mobile node that is still needed in the network gets a low battery threshold, a mobile charger can be requested and the mobile node will stay at its current position and at the same time be charged, hence continue assisting the network's needs. In the case that currently no mobile charger is available, the mobile node would be returning to its initial position and its place would be taken by another mobile node, as our current algorithms do. As a result, in both approaches, the recovery of a low-energy node will be able to extend the lifetime of the network and provide a solution of the energy constrain in WSNs.

Initial Placement of Mobile Nodes

Another important open question for mobility in WSNs is the initial placement of the mobile nodes. It would be interesting to investigate different approaches in terms of different parameters that will define the initial placement of the mobile nodes in the network.

The placement plan can be divided into two categories: the mobile nodes as a whole group or the mobile nodes divided into groups. In the first category, all mobile nodes are used as a group and are placed at the same initial location, whereas, in the second category, the mobile nodes are divided into groups which are placed into different areas of the network, as their initial position.

Another part that needs to be defined are some parameters that play a significant role in the selection of the initial placement. These parameters could be the following:

The problem that needs to be resolved. Based on the problem that has occurred in the network, the solution provided needs to be different. Each solution is a different approach in resolving the problem, which can be also a different direction for the initial placement of the mobile nodes. For example, in a jamming attack the solution needs to overcome the affected area, whereas in a fault problem the solution can be directly in the affected area. In the first case, the mobile node needs to be placed in a position that is not in the reach of the attack, which means outside of the affected area of the attacker. On the other hand, when a faulty node exists in the network, in terms of energy exhaustion, a simple replacement is all needed to resolve the problem.

The detection mechanism used. When a problem is detected by the sink node, so a centralized approached is used, it is needed that the mobile nodes are in the reach of the sink node's communication range. The sink node is the one that will make the computations and take actions, which then will be forwarded to the mobile nodes. On the other hand, in a decentralized approach, where the detection is done by any neighboring node of a faulty node, it may be better to place groups of mobile nodes in the network. For example, when the detection is done by the sink node it is helpful for the mobile nodes to be placed near the sink so that the communication between them is quicker. However, in case the detection is done by the neighboring nodes of faulty nodes, placing in different locations of the network small groups of mobile nodes gives the opportunity to the detection node to communicate directly with the mobile node, which results in smaller energy and time consumption.

The topology of the network. The nature of the topology of a network can also be important for the initial placement of the mobile nodes. In some scenarios it would be best to divide the mobile nodes into groups and scatter them in the network. On the other hand, in other scenarios all mobile nodes would be best to be grouped only in one position. For example, in a ring topology it is better to initially placed the mobile nodes in the center in such a position where it minimizes the cost of moving to a new position. However, in a tree topology the mobile nodes may be placed near the root, meaning the sink node.

Real Deployment and Evaluation

An important direction of the research in WSNs is to expand the evaluation method of these algorithms by being experimented in a real sensor setting. Most algorithms are evaluated in simulations, but the results shown may not be accurate when they are applied in real environments. For accurate and unequivocal results the evaluation method should become more experimental by deploying and evaluating their performance using real sensors executing in a real life environment.

The main reason we have used Contiki OS to implement our algorithms is because it emulates the behavior of a real network. Furthermore, the programs are readily deployable to a real sensor environment with minimal adjustments. In particular, the code developed for the simulation can easily be exported and uploaded to a real sensor, leading to a real deployment. Therefore, part of the future work is to deploy and evaluate our proposed algorithmic solutions in a real sensor setting. There are many options on doing so. One case is to use real IoT testbeds, such as FIT IoTLab [2], which is a large-scale real testbed platform within the OpenLab Experimental Facility. The FIT IoT-Lab has six sites across France that include in total 2700 wireless sensor nodes. Another case is to use a small scale IoT testbed at the University of Cyprus premises using real nodes, such as Raspberry Pi, to examine obstacles that were not anticipated and were not revealed using COOJA.

Bibliography

[1] G. H. Adday, S. K. Subramaniam, Z. A. Zukarnain, and N. Samian. Fault tolerance structures in wireless sensor networks (wsns): Survey, classification, and future directions. *Sensors*, 22(16):6041, 2022.

[2] C. Adjih, E. Baccelli, E. Fleury, G. Harter, N. Mitton, T. Noel, R. Pissard-Gibollet, F. Saint-Marcel, G. Schreiner, J. Vandaele, et al. Fit iot-lab: A large scale open experimental iot testbed. In *2015 IEEE 2nd World Forum on Internet of Things (WF-IoT)*, pages 459–464. IEEE, 2015.

[3] V. Agarwal, S. Tapaswi, and P. Chanak. A survey on path planning techniques for mobile sink in iot-enabled wireless sensor networks. *Wireless Personal Communications*, pages 1–28, 2021.

[4] V. K. Akram, O. Dagdeviren, and B. Tavli. Distributed k-connectivity restoration for fault tolerant wireless sensor and actuator networks: Algorithm design and experimental evaluations. *IEEE Trans. Reliab.*, 70(3):1112–1125, 2021.

[5] M. Anuradha, A. Swetha, and M. Doraipandian. Fault node detection and connectivity restoration with mobile relay node in wireless sensor networks.

[6] C. Balasubramanian, M. Maragatharajan, and S. P. Balakannan. Inventive approach of path planning mechanism for mobile anchors in WSN. *J. Ambient Intell. Humaniz. Comput.*, 12(3):3959–3967, 2021.

[7] R. Beulah Jayakumari and V. Jawahar Senthilkumar. Priority based congestion control dynamic clustering protocol in mobile wireless sensor networks. *The Scientific World Journal*, 2015, 2015.

[8] A. Boukerche, D. Efstathiou, and S. Nikoletseas. Direction-based adaptive data propagation for heterogeneous sensor mobility. *Journal of Parallel and Distributed Computing*, 72(6):778–790, 2012.

[9] S. Chouikhi, I. El Korbi, Y. Ghamri-Doudane, and L. A. Saidane. A survey on fault tolerance in small and large scale wireless sensor networks. *Computer Communications*, 69:22–37, 2015.

[10] I. Doroftei, V. Grosu, and V. Spinu. Design and control of an omni-directional mobile robot. In *Novel algorithms and techniques in telecommunications, automation and industrial electronics*, pages 105–110. Springer, 2008.

[11] M. El Fissaoui, A. Beni-hssane, S. Ouhmad, and K. El Makkaoui. A survey on mobile agent itinerary planning for information fusion in wireless sensor networks. *Archives of Computational Methods in Engineering*, pages 1–12, 2020.

[12] L. Farzinvash, S. Najjar-Ghabel, and T. Javadzadeh. A distributed and energy-efficient approach for collecting emergency data in wireless sensor networks with mobile sinks. *AEU-International Journal of Electronics and Communications*, 108:79–86, 2019.

[13] S. Feng, H. Shi, L. Huang, S. Shen, S. Yu, H. Peng, and C. Wu. Unknown hostile environment-oriented autonomous WSN deployment using a mobile robot. *J. Netw. Comput. Appl.*, 182:103053, 2021.

[14] M. D. Francesco, S. K. Das, and G. Anastasi. Data collection in wireless sensor networks with mobile elements: A survey. *TOSN*, 8(1):7:1–7:31, 2011.

[15] S. Gandham, M. Dawande, R. Prakash, and S. Venkatesan. Energy efficient schemes for wireless sensor networks with multiple mobile base stations. In *Proceedings of the Global Telecommunications Conference, 2003. GLOBECOM*, pages 377–381, 2003.

[16] Y. Gu, F. Ren, Y. Ji, and J. Li. The evolution of sink mobility management in wireless sensor networks: A survey. *IEEE Communications Surveys and Tutorials*, 18(1):507–524, 2016.

[17] K. Gulati, R. S. K. Boddu, D. Kapila, S. L. Bangare, N. Chandnani, and G. Saravanan. A review paper on wireless sensor network techniques in internet of things (iot). *Materials Today: Proceedings*, 2021.

[18] E. B. Hamida and G. Chelius. A line-based data dissemination protocol for wireless sensor networks with mobile sink. In *Proceedings of IEEE International Conference on Communications, ICC*, pages 2201–2205, 2008.

[19] L. Hou, L. Zhang, and J. Kim. Energy modeling and power measurement for mobile robots. *Energies*, 12(1):27, 2019.

[20] T. Hou and V. Li. Transmission range control in multihop packet radio networks. *IEEE Trans. Communications*, 34(1):38–44, 1986.

[21] H. Huang, C. Huang, and D. Ma. The cluster based compressive data collection for wireless sensor networks with a mobile sink. *AEU-International Journal of Electronics and Communications*, 108:206–214, 2019.

[22] H. Huang, A. V. Savkin, M. Ding, and C. Huang. Mobile robots in wireless sensor networks: A survey on tasks. *Computer Networks*, 148:1–19, 2019.

[23] C. Ioannou and V. Vassiliou. An intrusion detection system for constrained WSN and iot nodes based on binary logistic regression. In *Proceedings of the 21st ACM International Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems, MSWiM 2018, Montreal, QC, Canada, October 28 - November 02, 2018*, pages 259–263, 2018.

[24] C. Ioannou, V. Vassiliou, and C. Sergiou. RMT: A wireless sensor network monitoring tool. In *Proceedings of the 13th ACM Symposium on Performance Evaluation of Wireless Ad Hoc, Sensor, & Ubiquitous Networks, PE-WASUN 2016, Malta, November 13-17, 2016*, pages 45–49, 2016.

[25] C. Ioannou, V. Vassiliou, and C. Sergiou. An intrusion detection system for wireless sensor networks. In *24th International Conference on Telecommunications, ICT 2017, Limassol, Cyprus, May 3-5, 2017*, pages 1–5, 2017.

[26] A. E. Irish, S. Terence, and J. Immaculate. Efficient data collection using dynamic mobile sink in wireless sensor network. In *Wireless Communication Networks and Internet of Things*, pages 141–149. Springer, 2019.

[27] M. F. Jaramillo-Morales, S. Dogru, J. B. Gomez-Mendoza, and L. Marques. Energy estimation for differential drive mobile robots on straight and rotational trajectories. *International Journal of Advanced Robotic Systems*, 17(2):1729881420909654, 2020.

[28] D. Kandris, C. Nakas, D. Vomvas, and G. Koulouras. Applications of wireless sensor networks: an up-to-date survey. *Applied System Innovation*, 3(1):14, 2020.

[29] K. Karenos and V. Kalogeraki. Facilitating congestion avoidance in sensor networks with a mobile sink. In *Proceedings of the 28th IEEE Real-Time Systems Symposium (RTSS*, pages 321–332, 2007.

[30] R. Katsuma, Y. Murata, N. Shibata, K. Yasumoto, and M. Ito. Extending k-coverage lifetime of wireless sensor networks using mobile sensor nodes. In *5th IEEE International Conference on Wireless and Mobile Computing, Networking and Communications, WiMob*, pages 48–54, 2009.

[31] M. I. Khan, W. N. Gansterer, and G. Haring. Congestion avoidance and energy efficient routing protocol for wireless sensor networks with a mobile sink. *JNW*, 2(6):42–49, 2007.

[32] A. Kinalis, S. Nikoletseas, D. Patroumpa, and J. Rolim. Biased sink mobility with adaptive stop times for low latency data collection in sensor networks. *Information fusion*, 15:56–63, 2014.

[33] D. Knuth. The art of computer programming: Generating all combinations and partitions (vol. 4, fascicle 3), 2005.

[34] K. Koosheshi and S. Ebadi. Optimization energy consumption with multiple mobile sinks using fuzzy logic in wireless sensor networks. *Wirel. Networks*, 25(3):1215–1234, 2019.

[35] M. Koutroullos, C. Sergiou, and V. Vassiliou. Mobile-CC: Introducing Mobility to WSNs for Congestion Mitigation in Heavily Congested Areas. In *Telecommunications (ICT), 2011 18th International Conference on*, pages 400 –405, May 2011.

[36] M. Koutroullos, C. Sergiou, and V. Vassiliou. Mobile-cc: Introducing mobility to wsns for congestion mitigation in heavily congested areas. In *18th International Conference on Telecommunications, ICT 2011, Ayia Napa, Cyprus, May 8-11, 2011*, pages 400–405, 2011.

[37] D. Koutsonikolas, S. M. Das, and Y. C. Hu. Path planning of mobile landmarks for localization in wireless sensor networks. *Comput. Commun.*, 30(13):2577–2592, 2007.

[38] J. F. Kurose and K. W. Ross. *Computer networking - a top-down approach featuring the internet.* Addison-Wesley-Longman, 2001.

[39] S. Latambale and S. Sirsikar. A survey of various sink mobility based techniques in wireless sensor network. In *Proceedings of the ACM Symposium on Women in Research 2016*, pages 45–50, 2016.

[40] H. Lee, J. Lee, S. Kim, and S. Noh. Region based data dissemination scheme for mobile sink groups in wireless sensor networks. In *Proceedings of the Global Communications Conference, 2010. GLOBECOM 2010, 6-10 December 2010, Miami, Florida, USA*, pages 1–5, 2010.

[41] J. Lee, S. Oh, S. Park, Y. Yim, S. Kim, and E. Lee. Active data dissemination for mobile sink groups in wireless sensor networks. *Ad Hoc Networks*, 72:56–67, 2018.

[42] W. Liang, J. Luo, and X. Xu. Prolonging network lifetime via a controlled mobile sink in wireless sensor networks. In *Proceedings of the Global Communications Conference, 2010. GLOBECOM*, pages 1–6, 2010.

[43] C. Lin, P. Chou, and C. Chou. HCDD: hierarchical cluster-based data dissemination in wireless sensor networks with mobile sink. In *Proceedings of the International Conference on Wireless Communications and Mobile Computing, IWCMC*, pages 1189–1194, 2006.

[44] S. Lloyd. Least squares quantization in pcm. *IEEE Transactions on Information Theory*, 28(2):129–137, 1982.

[45] A. Madhja, S. E. Nikoletseas, and T. P. Raptis. Distributed wireless power transfer in sensor networks with multiple mobile chargers. *Computer Networks*, 80:89–108, 2015.

[46] A. Madhja, S. E. Nikoletseas, and T. P. Raptis. Hierarchical, collaborative wireless energy transfer in sensor networks with multiple mobile chargers. *Computer Networks*, 97:98–112, 2016.

[47] A. Madhja, S. E. Nikoletseas, and A. A. Voudouris. Mobility-aware, adaptive algorithms for wireless power transfer in ad hoc networks. In *Algorithms for Sensor Systems - 14th International Symposium on Algorithms and Experiments for Wireless Sensor Networks, ALGOSENSORS 2018, Helsinki, Finland, August 23-24, 2018, Revised Selected Papers*, pages 145–158, 2018.

[48] A. Madhja, S. E. Nikoletseas, and A. A. Voudouris. Adaptive wireless power transfer in mobile ad hoc networks. *Computer Networks*, 152:87–97, 2019.

[49] S. Maurya, V. K. Jain, and D. R. Chowdhury. Delay aware energy efficient reliable routing for data transmission in heterogeneous mobile sink wireless sensor network. *J. Netw. Comput. Appl.*, 144:118–137, 2019.

[50] N. Mazumdar, S. Roy, A. Nag, and S. Nandi. An adaptive hierarchical data dissemination mechanism for mobile data collector enabled dynamic wireless sensor network. *Journal of Network and Computer Applications*, 186:103097, 2021.

[51] Y. Mei, C. Xian, S. Das, Y. C. Hu, and Y.-H. Lu. Repairing sensor network using mobile robots. In *Workshop on Wireless Ad hoc and Sensor Networks*, 2006.

[52] H. Mo, E. Lee, S. Park, and S. Kim. Virtual line-based data dissemination for mobile sink groups in wireless sensor networks. *IEEE Communications Letters*, 17(9):1864–1867, 2013.

[53] E. Moridi, M. Haghparast, M. Hosseinzadeh, and S. J. Jassbi. Fault management frameworks in wireless sensor networks: A survey. *Computer communications*, 155:205–226, 2020.

[54] T. Muhammed and R. A. Shaikh. An analysis of fault detection strategies in wireless sensor networks. *J. Netw. Comput. Appl.*, 78:267–287, 2017.

[55] L. Nguyen and H. T. Nguyen. Mobility based network lifetime in wireless sensor networks: A review. *Comput. Networks*, 174:107236, 2020.

[56] A. Nicolaou, N. Temene, C. Sergiou, C. Georgiou, and V. Vassiliou. Utilizing mobile nodes for congestion control in wireless sensor networks. In *15th International Conference on Distributed Computing in Sensor Systems, DCOSS 2019, Santorini, Greece, May 29-31, 2019*, pages 176–178. IEEE, 2019.

[57] S. E. Nikoletseas, T. P. Raptis, and C. Raptopoulos. Wireless charging for weighted energy balance in populations of mobile peers. *Ad Hoc Networks*, 60:1–10, 2017.

[58] C. OS. Contiki: The open source os for the internet of things. `http://www.contiki-os.org/`.

[59] A. Pang, F. Chao, H. Zhou, and J. Zhang. The method of data collection based on multiple mobile nodes for wireless sensor network. *IEEE Access*, 8:14704–14713, 2020.

[60] F. Papi and H. Barati. Hdrm: A hole detection and recovery method in wireless sensor network. *International Journal of Communication Systems*, 35(8):e5120, 2022.

[61] S. Park, E. Lee, M. Jin, and S. Kim. Novel strategy for data dissemination to mobile sink groups in wireless sensor networks. *IEEE Communications Letters*, 14(3):202–204, 2010.

[62] S. Park, E. Lee, H. Park, H. Lee, and S. Kim. Mobile geocasting to support mobile sink groups in wireless sensor networks. *IEEE Communications Letters*, 14(10):939–941, 2010.

[63] L. Payá, A. Gil, O. Reinoso, M. Juliá, L. Riera, and L. Jiménez. Distributed platform for the control of the wifibot robot through internet. *IFAC Proceedings Volumes*, 39(6):59–64, 2006.

[64] S. Prakash and V. Saroj. A review of wireless charging nodes in wireless sensor networks. *Data science and big data analytics*, pages 177–188, 2019.

[65] V. S. Rao and M. Dakshayini. An sdn-based strategy for reliable data transmission in mobile wireless sensor networks. In *EAI International Conference on Big Data Innovation for Sustainable Cognitive Computing*, pages 87–96. Springer, 2020.

[66] S. Raza, L. Wallgren, and T. Voigt. SVELTE: real-time intrusion detection in the internet of things. *Ad Hoc Networks*, 11(8):2661–2674, 2013.

[67] A. P. Renold and B. G. Athi. Energy efficient secure data collection with path-constrained mobile sink in duty-cycled unattended wireless sensor network. *Pervasive and Mobile Computing*, 55:1–12, 2019.

[68] J. Rezazadeh. Mobile wireles sensor networks overview. *International Journal of Computer Communications and Networks (IJCCN)*, 2(1), 2012.

[69] K. Römer and F. Mattern. The design space of wireless sensor networks. *IEEE Wireless Commun.*, 11(6):54–61, 2004.

[70] C. Sergiou, P. Antoniou, and V. Vassiliou. A comprehensive survey of congestion control protocols in wireless sensor networks. *IEEE Communications Surveys Tutorials*, 16(4):1839–1859, 2014.

[71] C. Sergiou and V. Vassiliou. Estimating maximum traffic volume in wireless sensor networks using fluid dynamics principles. *IEEE Commun. Lett.*, 17(2):257–260, 2013.

[72] C. Sergiou, V. Vassiliou, and A. Paphitis. Congestion Control in Wireless Sensor Networks through Dynamic Alternative Path Selection. 75, Part A:226 – 238, 2014.

[73] S. Sharma, R. K. Bansal, and S. Bansal. Issues and challenges in wireless sensor networks. In *2013 International Conference on Machine Intelligence and Research Advancement*, pages 58–62. IEEE, 2013.

[74] J. Sheu, K. Hsieh, and P. Cheng. Design and implementation of mobile robot for nodes replacement in wireless sensor networks. *J. Inf. Sci. Eng.*, 24(2):393–410, 2008.

[75] S. K. Singh and P. Kumar. A comprehensive survey on trajectory schemes for data collection using mobile elements in wsns. *J. Ambient Intell. Humaniz. Comput.*, 11(1):291–312, 2020.

[76] J. Sumathi and R. L. Velusamy. A review on distributed cluster based routing approaches in mobile wireless sensor networks. *J. Ambient Intell. Humaniz. Comput.*, 12(1):835–849, 2021.

[77] H. Takagi and L. Kleinrock. Optimal transmission ranges for randomly distributed packet radio terminals. *IEEE Trans. Communications*, 32(3):246–257, 1984.

[78] T. Toledo and H. N. Koutsopoulos. Statistical validation of traffic simulation models. *Transportation Research Record*, 1876(1):142–150, 2004.

[79] A. S. Toor and A. Jain. Energy aware cluster based multi-hop energy efficient routing protocol using multiple mobile nodes (meacbm) in wireless sensor networks. *AEU-International Journal of Electronics and Communications*, 102:41–53, 2019.

[80] T. T. Truong, K. N. Brown, C. J. Sreenan, and T. T. Truong. Using mobile sinks in wireless sensor networks to improve building emergency response. In *Royal Irish Academy Research Colloquium on Wireless as an Enabling Technology.* Royal Irish Academy, 2010.

[81] C. Tunca, S. Isik, M. Y. Donmez, and C. Ersoy. Distributed mobile sink routing for wireless sensor networks: A survey. *IEEE Communications Surveys and Tutorials*, 16(2):877–897, 2014.

[82] M. Vecchio and R. López-Valcarce. Improving area coverage of wireless sensor networks via controllable mobile nodes: A greedy approach. *J. Network and Computer Applications*, 48:1–13, 2015.

[83] K. L. Wagstaff. *Constrained Clustering*, pages 220–221. Springer US, 2010.

[84] M. Wahab, F. Rios-Gutierrez, and A. El Shahat. *Energy modeling of differential drive robots*. IEEE, 2015.

[85] J. Wang, Y. Gao, W. Liu, A. K. Sangaiah, and H. Kim. Energy efficient routing algorithm with mobile sink support for wireless sensor networks. *Sensors*, 19(7):1494, 2019.

[86] J. Wang, Y. Gao, W. Liu, A. K. Sangaiah, and H. Kim. An intelligent data gathering schema with data fusion supported for mobile sink in wireless sensor networks. *IJDSN*, 15(3), 2019.

[87] J. Wang, Y. Gao, X. Yin, F. Li, and H. Kim. An enhanced PEGASIS algorithm with mobile sink support for wireless sensor networks. *Wireless Communications and Mobile Computing*, 2018:9472075:1–9472075:9, 2018.

[88] J. Wang, Y. Gao, X. Yin, F. Li, and H.-J. Kim. An enhanced pegasis algorithm with mobile sink support for wireless sensor networks. *Wireless Communications and Mobile Computing*, 2018, 2018.

[89] Y. Wang, K. Huang, P. Fu, and J. Wang. Mobile sink routing protocol with registering in cluster-based wireless sensor networks. In *Ubiquitous Intelligence and Computing, 5th International Conference, UIC*, pages 352–362, 2008.

[90] M. Wu. An efficient hole recovery method in wireless sensor networks. In *2022 24th International Conference on Advanced Communication Technology (ICACT)*, pages 1399–1404. IEEE, 2022.

[91] R. Yarinezhad. Reducing delay and prolonging the lifetime of wireless sensor network using efficient routing protocol based on mobile sink and virtual infrastructure. *Ad Hoc Networks*, 84:42–55, 2019.

[92] C. W. Yu, E. Chen, and C.-C. Fang. Deploying mobile nodes to connect wireless sensor networks using novel algorithms. In *Wireless Algorithms, Systems and Applications*, pages 199–204, 2007.

[93] Y.-G. Yue and P. He. A comprehensive survey on the reliability of mobile wireless sensor networks: Taxonomy, challenges, and future directions. *Information Fusion*, 44:188–204, 2018.

[94] H. Zhao, S. Guo, X. Wang, and F. Wang. Energy-efficient topology control algorithm for maximizing network lifetime in wireless sensor networks with mobile sink. *Appl. Soft Comput.*, 34:539–550, 2015.

[95] P. Zhong and F. Ruan. An energy efficient multiple mobile sinks based routing algorithm for wireless sensor networks. In *IOP Conference Series: Materials Science and Engineering*, volume 323, page 012029, 2018.

[96] D. Zorbas and T. Razafindralambo. Modeling the power consumption of a wifibot and studying the role of communication cost in operation time. *CoRR*, abs/1512.04380, 2015.

Chapter A

Algorithm Flowcharts

In this section we present the flowcharts of each algorithm mentioned in the paper. These flowcharts show a simpler version of the algorithm presenting the higher level idea.

A.1 Dynamic MobileCC Flowchart

In this section we present the flowchart of the Dynamic MobileCC algorithm (see Fig. A.1) presented in Section 3.1.

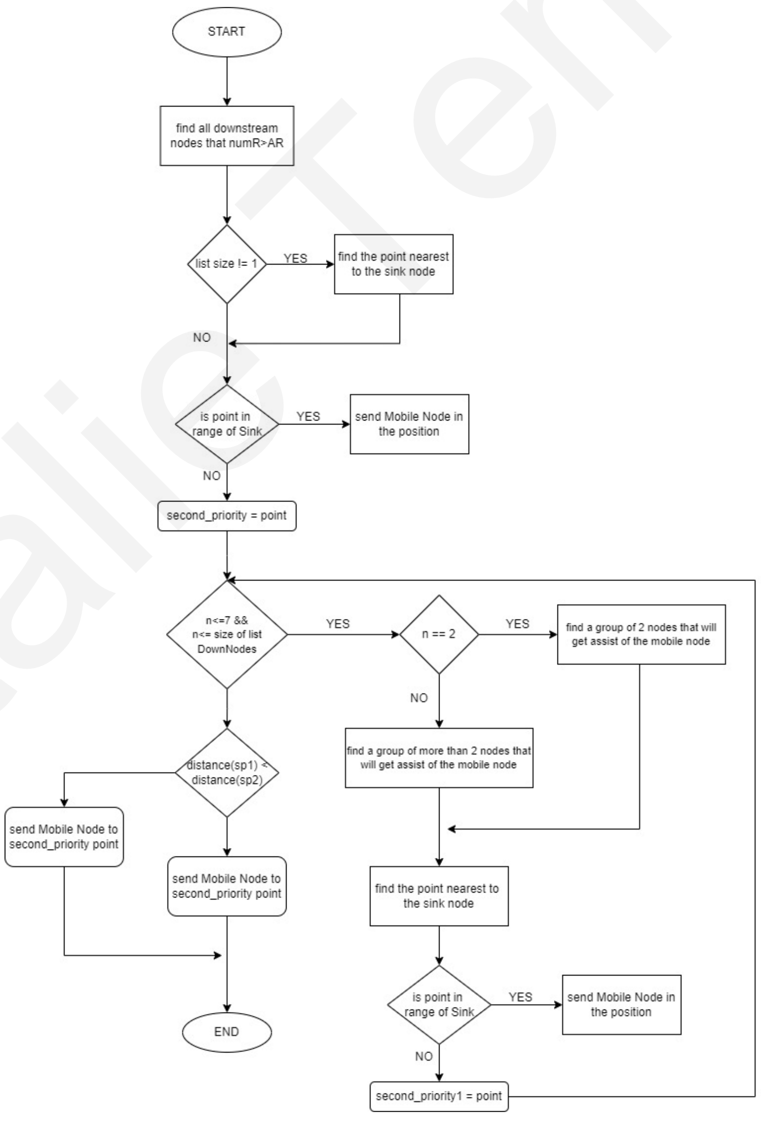

START
find all downstream nodes that numR>AR
list size != 1
YES
find the point nearest to the sink node
NO
is point in range of Sink
YES
send Mobile Node in the position
NO
second_priority = point
n<=7 &&
n<= size of list DownNodes
YES
n == 2
YES
find a group of 2 nodes that will get assist of the mobile node
NO
find a group of more than 2 nodes that will get assist of the mobile node
distance(sp1) < distance(sp2)
send Mobile Node to second_priority point
send Mobile Node to second_priority point
find the point nearest to the sink node
is point in range of Sink
YES
send Mobile Node in the position
NO
second_priority1 = point
END

Figure A.1: Dynamic MobileCC Flowchart

A.2 Direct MobileCC Algorithm Flowchart

In this section we present the flowchart of the Direct MobileCC algorithm (see Fig. A.2) presented in Section 3.2.

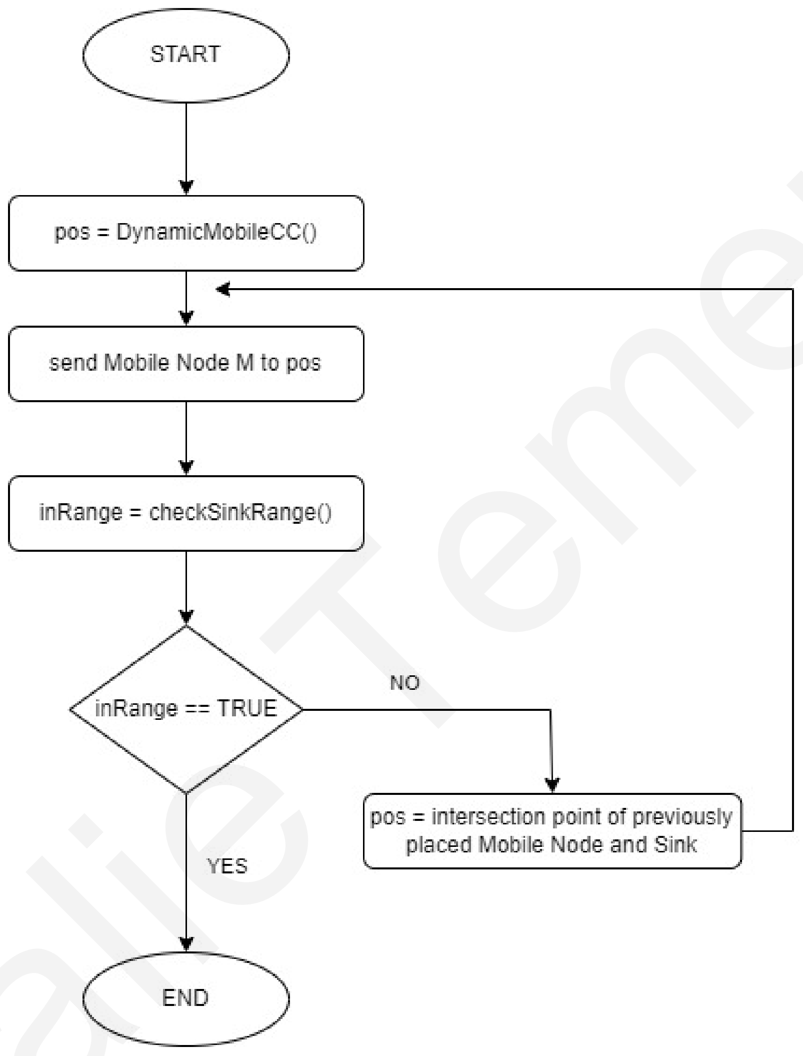

START
pos = DynamicMobileCC()
send Mobile Node M to pos
inRange = checkSinkRange()
inRange == TRUE
NO
pos = intersection point of previously placed Mobile Node and Sink
YES
END

Figure A.2: Direct MobileCC Flowchart

A.3 Energy-aware Node Placement Algorithm Flowchart

In this section we present the flowchart of the Energy-aware Node Placement Algorithm (see Fig. A.3) presented in Chapter 4.

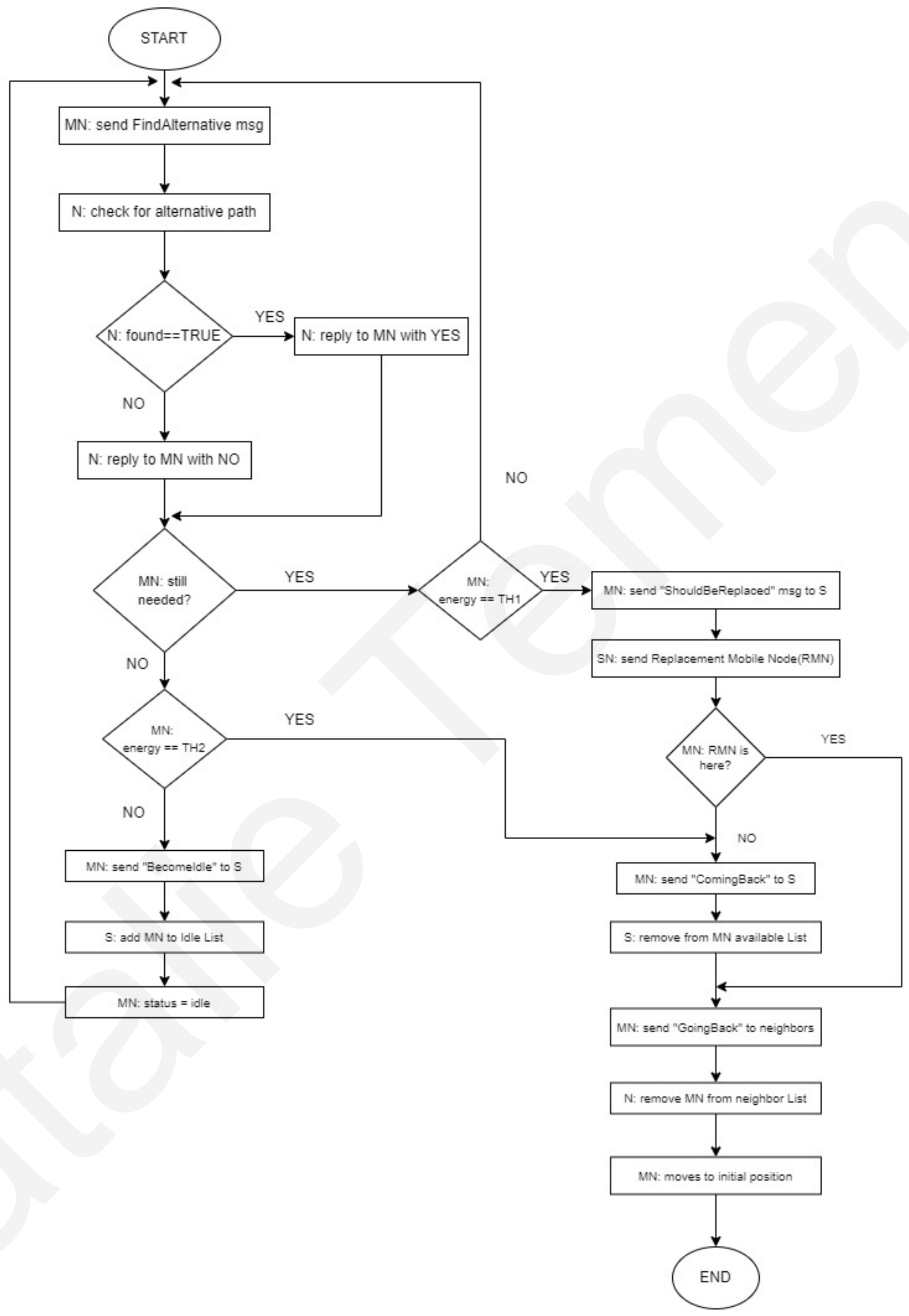

START
MN: send FindAlternative msg
N: check for alternative path
N: found==TRUE
YES
N: reply to MN with YES
NO
N: reply to MN with NO
NO
MN: still needed?
YES
MN: energy == TH1
YES
MN: send "ShouldBeReplaced" msg to S
SN: send Replacement Mobile Node(RMN)
NO
MN: energy == TH2
YES
MN: RMN is here?
YES
NO
NO
MN: send "BecomeIdle" to S
S: add MN to Idle List
MN: status = idle
MN: send "ComingBack" to S
S: remove from MN available List
MN: send "GoingBack" to neighbors
N: remove MN from neighbor List
MN: moves to initial position
END

Figure A.3: Energy-aware Node Placement Algorithm Flowchart

A.4 Carrier-based Node Placement Algorithm Flowchart

In this section we present the flowchart of the Carrier-based Node Placement Algorithm (see Fig. A.4) presented in Chapter 5.

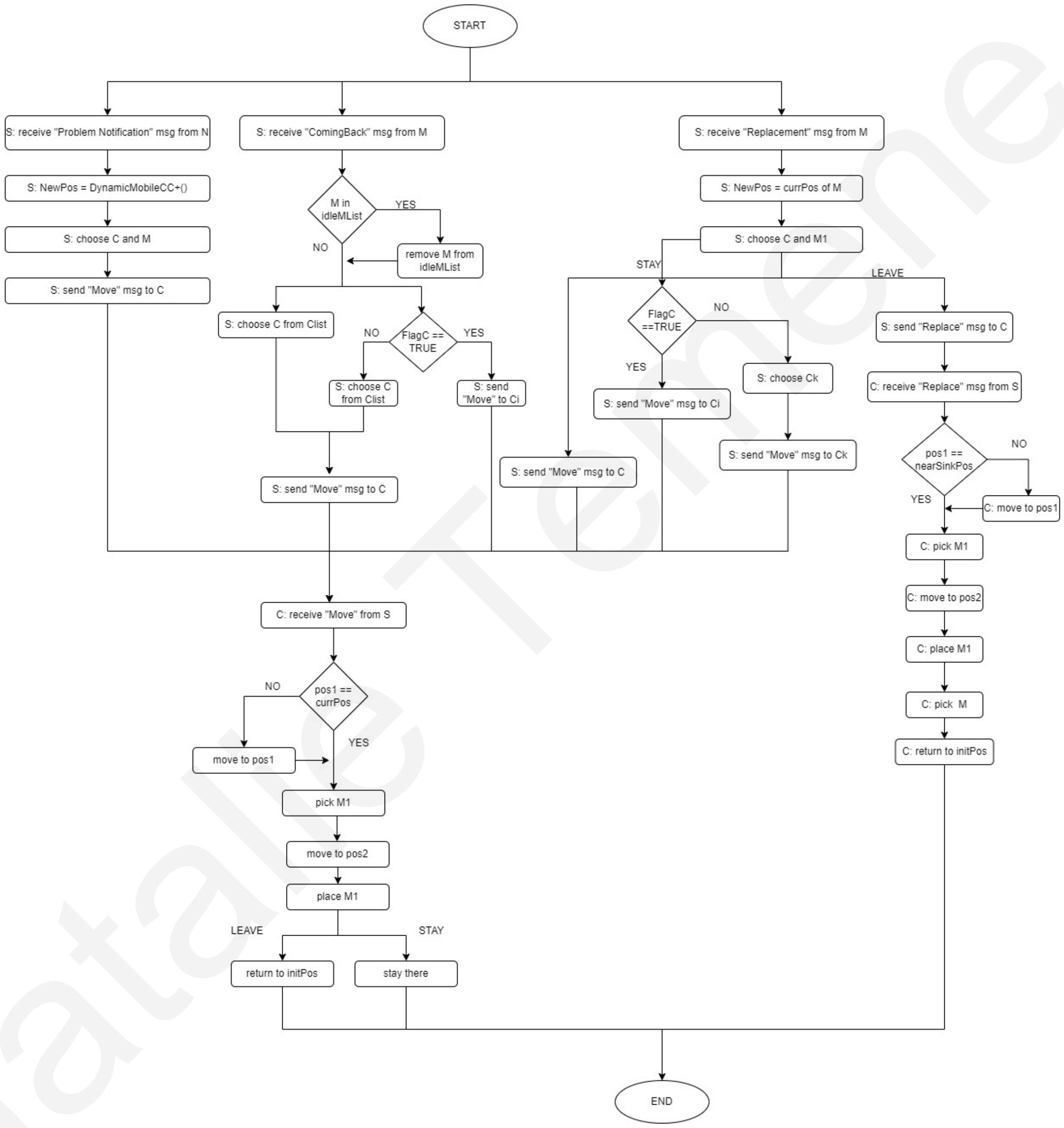


Figure A.4: Carrier-based Node Placement Algorithm Flowchart

A.5 Decentralized Fault Tolerant Node Placement Algorithm Flowchart

In this section we present the flowchart of the Decentralized Fault Tolerant Node Placement Algorithm Flowchart (see Fig. A.5) presented in Section 7.2.1.

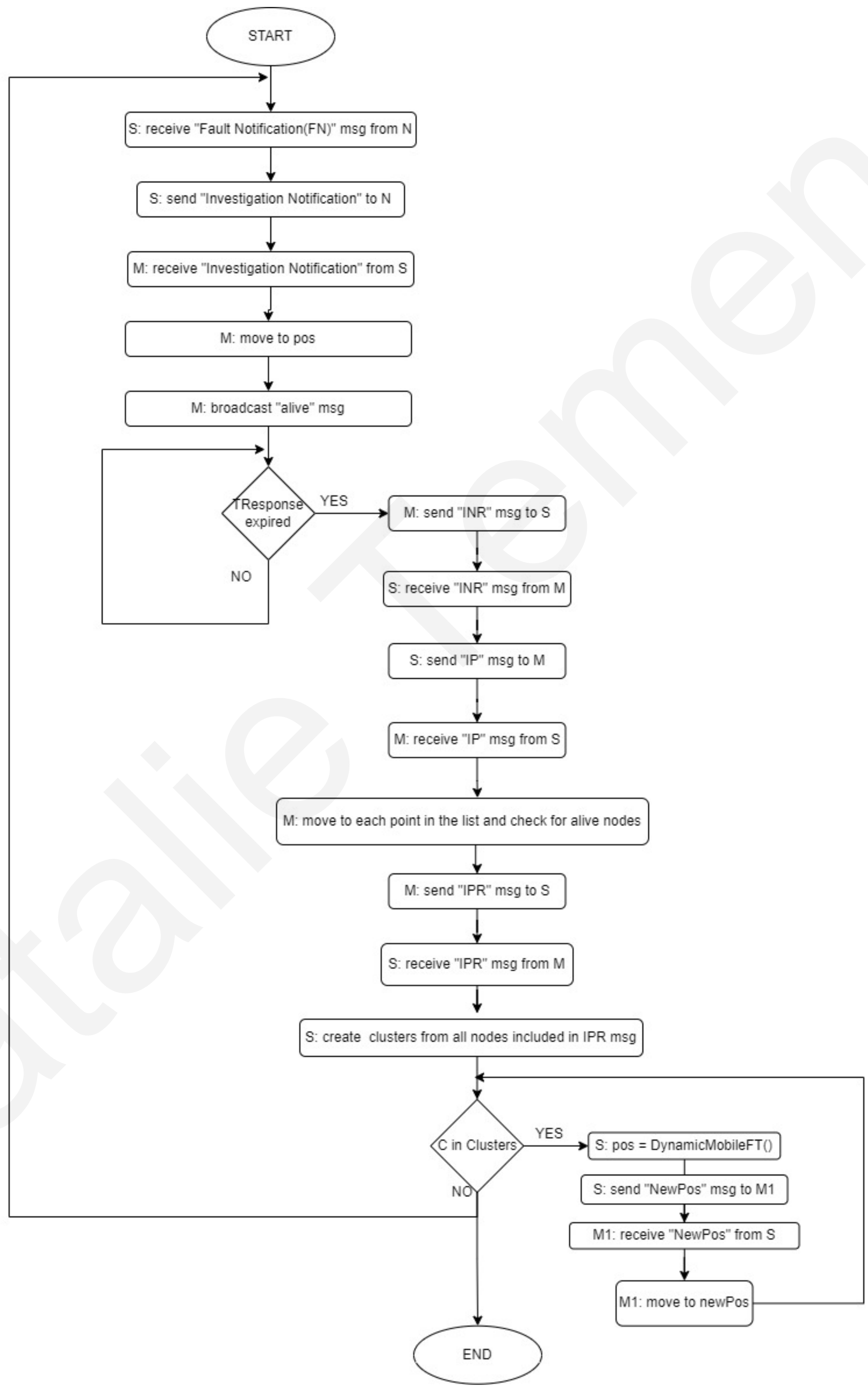

START
S: receive "Fault Notification(FN)" msg from N
S: send "Investigation Notification" to N
M: receive "Investigation Notification" from S
M: move to pos
M: broadcast "alive" msg
TResponse expired
YES
NO
M: send "INR" msg to S
S: receive "INR" msg from M
S: send "IP" msg to M
M: receive "IP" msg from S
M: move to each point in the list and check for alive nodes
M: send "IPR" msg to S
S: receive "IPR" msg from M
S: create clusters from all nodes included in IPR msg
C in Clusters
YES
NO
S: pos = DynamicMobileFT()
S: send "NewPos" msg to M1
M1: receive "NewPos" from S
M1: move to newPos
END

Figure A.5: Decentralized Fault Tolerant Node Placement Algorithm Flowchart

A.6 Centralized Fault Tolerant Node Placement Algorithm Flowchart

In this section we present the flowchart of the Decentralized Fault Tolerant Node Placement Algorithm Flowchart (see Fig. A.6) presented in Section 7.2.2.

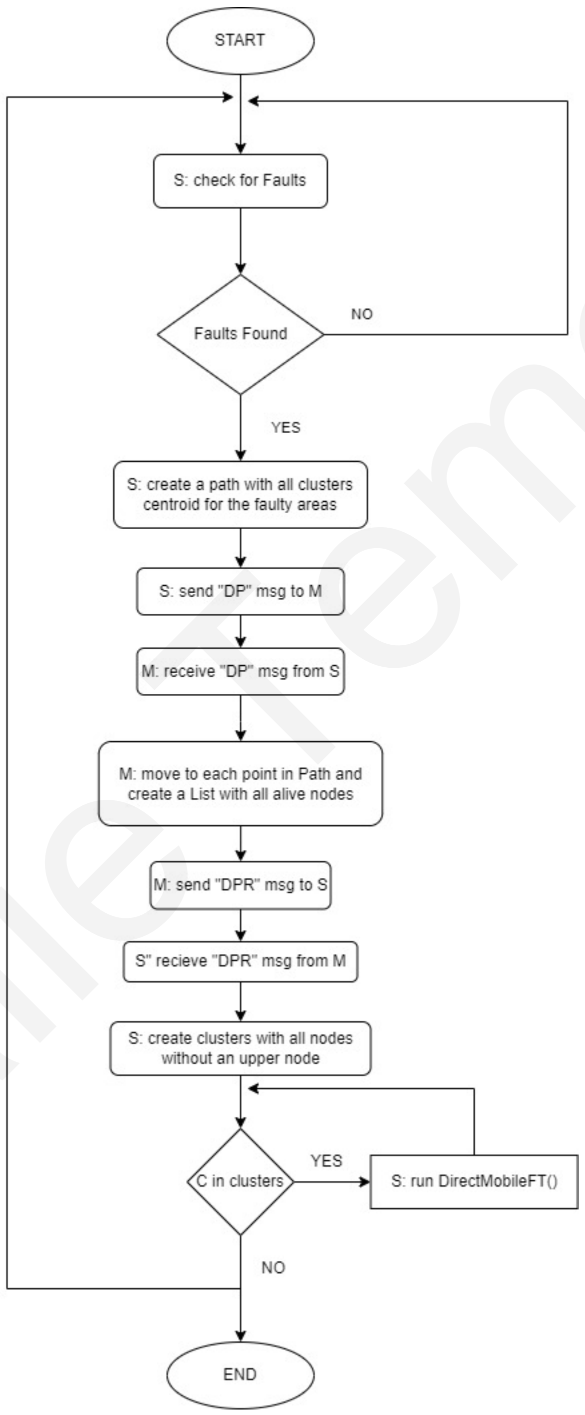

START
S: check for Faults
Faults Found
NO
YES
S: create a path with all clusters centroid for the faulty areas
S: send "DP" msg to M
M: receive "DP" msg from S
M: move to each point in Path and create a List with all alive nodes
M: send "DPR" msg to S
S" recieve "DPR" msg from M
S: create clusters with all nodes without an upper node
C in clusters
YES
S: run DirectMobileFT()
NO
END

Figure A.6: Centralized Fault Tolerant Node Placement Algorithm Flowchart

A.7 Dynamic MobileFT Flowchart

In this section we present the flowchart of the Dynamic MobileFT algorithm (see Fig. A.7) presented in Section 7.2.1.

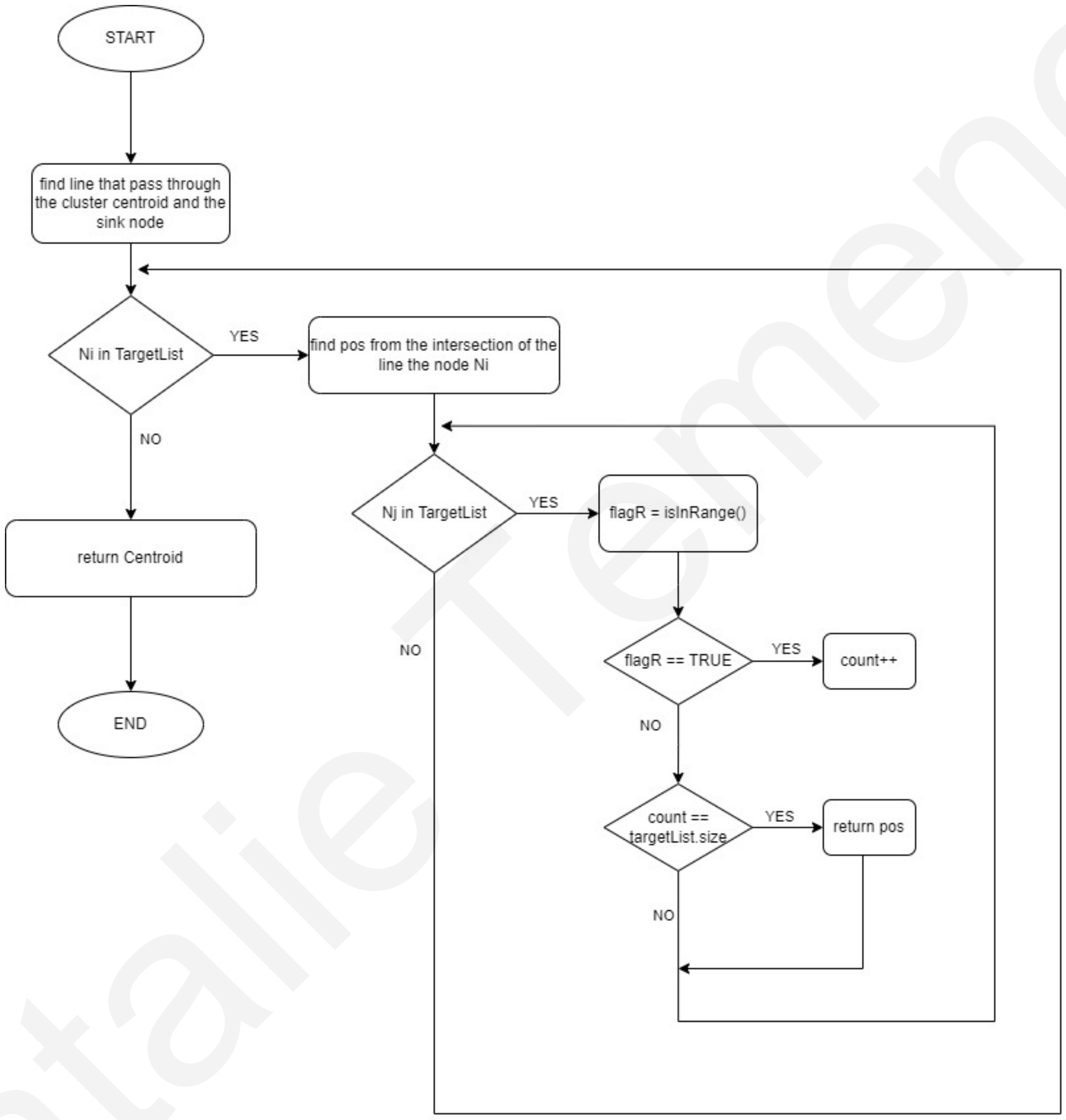


Figure A.7: Dynamic MobileFT Flowchart

A.8 Direct MobileFT Algorithm Flowchart

In this section we present the flowchart of the Direct MobileFT algorithm (see Fig. A.8) presented in Section 7.2.2.

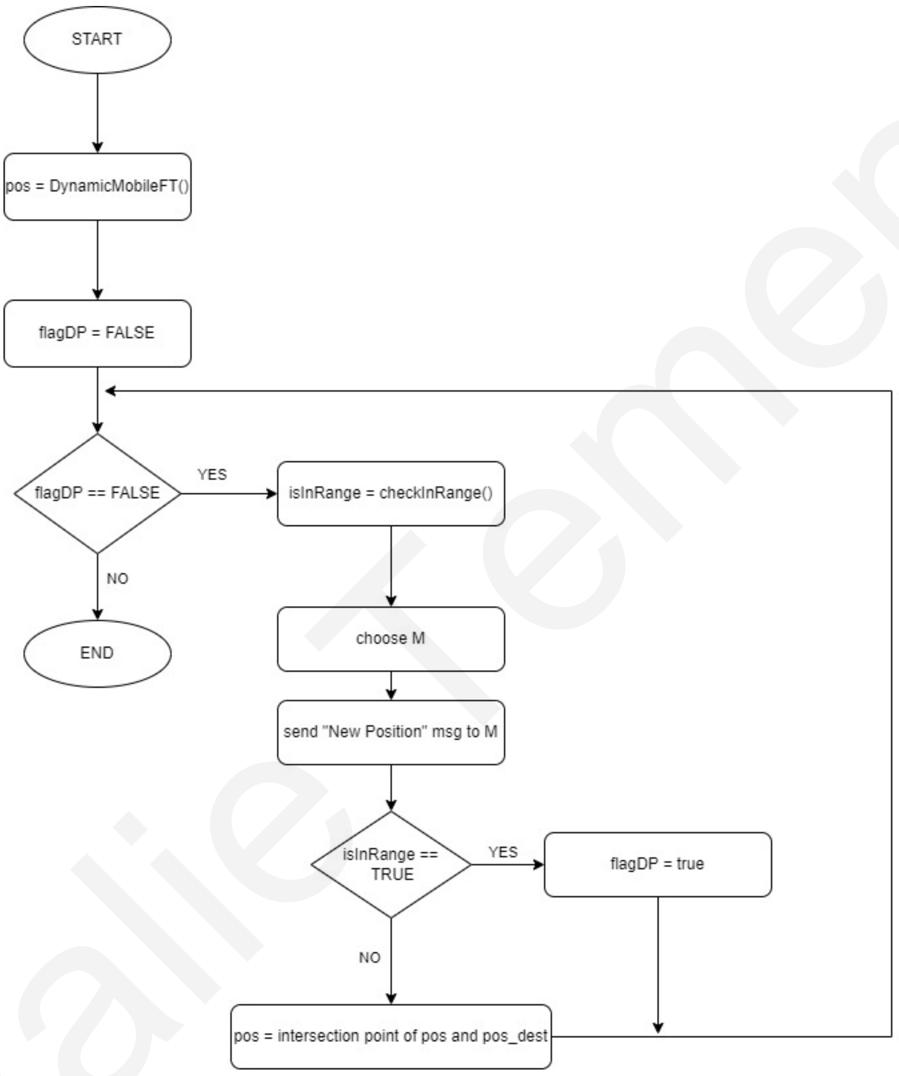


Figure A.8: Direct MobileFT Flowchart

Chapter B

Tables of Strengths and Limitations

In this section we present the tables of each paper mentioned in Chapter 2. The tables show the strengths and limitations of each algorithm.

B.1 Individual Mobile Sink

Table B.1 presents the strengths and limitations of each paper from Section 2.2.1.

Algorithm	Remarks	
Gandham et al. [15]	Strengths	relocation of the mobile sink at each round.
	Limitations	overhead in action to account node failure if done from itself.
Liang et al. [42]	Strengths	low computational complexity and high scalability
	Limitations	re-calculating the sojourn time in stage three.
Farzinvash et al. [12]	Strengths	low delay and different strategies for different priority data.
	Limitations	two methods of data collection done concurrently.
Khan et al. [31]	Strengths	clusters created by the mobile sink. The data determine the time of the mobile sink at the cluster head.
	Limitations	overhead at cluster head nodes.
Wang et al. [89]	Strengths	considers energy consumption and data dissemination inside and outside of the cluster.
	Limitations	overhead at the cluster and registration process.
Zhao et al. [94]	Strengths	used for large-scale networks. It uses only local information and considers the transmission range for energy.
	Limitations	
Zhong et al. [95]	Strengths	distributes evenly the hot spot nodes. Experiments with different numbers of mobile sinks.
	Limitations	at each round, a different cluster head based on the energy level. The path is not determined.
Wang et al. [86]	Strengths	data fussion with the use of neural networks.
	Limitations	overhead at cluster head nodes and agent nodes.
Wang et al. [85]	Strengths	the cluster head is only changed when energy level gets low.
	Limitations	overhead at the inter cluster communication process.
Maurya et al. [49]	Strengths	densley deployment large scale networks and different roles of the nodes.
	Limitations	overhead at the ring formation process.

Renold et al. [67]	Strengths	mobile sink determines the collection nodes. Secure data communication with cryptography.
	Limitations	mobile sink moves only on the periphery of the network. Overhead of the identification process.
Huang et al. [21]	Strengths	hierarchical communication. Implementation information.
	Limitations	overhead for the cluster head election method.
Lin et al. [43]	Strengths	light control overhead for data dissemination process.
	Limitations	overhead in sink localization registration.
Karenos et al. [29]	Strengths	it favors the good paths to avoid congestion.
	Limitations	continuously movement of the mobile sink in the network, the energy consumption is not account.
Hamida et al. [18]	Strengths	the rendez-vous line is placed in the middle of the network. Use of multiple sinks.
	Limitations	overhead at the data request process of the sink request.
Truong et al. [80]	Strengths	the data are stored until the reconnection of the network.
	Limitations	due to movement, the hop count changes frequently.
Wang et al. [87]	Strengths	use mobile sink to balance the energy consumption of the network and use threshold values to protect from energy exhaustion failure.
	Limitations	the energy of the mobile sink is not account, it is assumed unconstrained.
Irish et al. [26]	Strengths	the mobile sink visits the grid with the highest detection event frequency, meaning it provides priority.
	Limitations	overhead at cluster head nodes.

Table B.1: Strengths and Limitations of the Mobile Sinks Algorithms

B.2 Group of Mobile Sinks

Table B.2 presents the strengths and limitations of each paper from Section 2.2.1.

Algorithm	Remarks	
Park et al. [62]	Strengths	adopt the traditional geocasting method for a solution using mobile sink groups. Supports movement outside the group region without loosing the data dissemination.
	Limitations	overhead on location updates notifications in the network. The criteria of selecting the leader are not clarified. When the radius rate is increased, more flooding is needed which creates broadcasting problems.
Lee et al. [40]	Strengths	guarantees data transmission when the groups moves as a whole, as well as when each member sink moves inside or outside of the group region. Avoids frequent location updates from the member sinks to the leader or the source node.

	Limitations	the leader selection determination is not explained in the experimental method. Which criteria are followed based on the mission or the policy that define the leader.
Park et al. [61]	Strengths	Use of clusters to avoid leader selection. No dependency between the group and a leader which results in no overhead.
	Limitations	the access nodes and the cluster heads need to consume more energy for the data dissemination process. Overhead created from the cluster head selection process that uses the flooding method.
Mo et al. [52]	Strengths	A virtual line structure is used to storage data, without using the flooding method. The data are collected from mobile sinks from the line whenever needed. Creates an energy efficient solution based on the density of the nodes and the region size.
	Limitations	not explained how the selection of the leader is made, on which criteria. Overhead creating the group region.
Lee et al. [41]	Strengths	present three schemes based on the three mobility patterns that exist. Experiments with different number of mobile sinks in a group and different speed.
	Limitations	only evaluation of the random mobility pattern. The leader selection criteria are not defined.

Table B.2: Strengths and Limitations of the Mobile Sinks Group Algorithms

B.3 Mobile Nodes

Table B.3 presents the strengths and limitations of each paper from Section 2.2.2.

Algorithm	Remarks	
Mei et al. [51]	Strengths	Different solutions based on the location and behavior of the manager robot.
	Limitations	Fixed distributed solution: motion overhead, centralized solution: low scalability,two distributed solutions: high message cost.
Yu et al. [92]	Strengths	Use mobile nodes to connect a disconnected network.
	Limitations	No implementation details.
Sheu et al. [74]	Strengths	The mobile robot navigates without a map or location info, just with the receiving strength signal.
	Limitations	Navigates at wrong directions and does not consider the navigation path already created.
Katsuma et al. [30]	Strengths	Balances the communication traffic of the near sink nodes.
	Limitations	Yhe implementation difficulties of this solution are not mentioned.
Koutroullos et al. [36]	Strengths	Guarantees the delivery of the packets.
	Limitations	use of extra resources that its cost is not account.
Jayakumari et al. [7]	Strengths	Packets are classified based on their priority.

	Limitations	More energy consumption when adjusting the transmission route. The clusters change at each round, results in selecting new cluster heads at each round.
Vecchio et al. [82]	Strengths	Use of mobile nodes to increase the sensing area. Independent determination of the next position of the mobile nodes.
	Limitations	Evaluation only with five mobile nodes in the network.
Toor et al. [79]	Strengths	Uses a hybrid combination of clustering, mobility and multihoping.
	Limitations	Overhead in creating clusters and subclusters.
Rao et al. [65]	Strengths	One-hop transmission from the cluster head to the mobile node. Less number of retransmissions.
	Limitations	The cluster head selection criteria is not defined.
Pang et al. [59]	Strengths	Mobile nodes are used as sink nodes. The network is divided based on the number of the mobile nodes.
	Limitations	Evaluation only with three mobile nodes. Cluster heads get too much traffic, more possible to get faster energy exhausted.
Anuradha et al. (2020) [5]	Strengths	the energy level determination of failure of the node itself.
	Limitations	the periodic alert messages towards the getaway will get to much traffic in the network.
Bala Subramanian et al. [6]	Strengths	Use mobile anchor nodes that can localize the static nodes. Path planning with grid scan method.
	Limitations	Evaluation with only two mobile anchor nodes.
Feng et al [13]	Strengths	A mobile robot is self localized, it knwos the concurrent mapping, deployment and localization of the network.
	Limitations	Evaluation is performed with only three beacon nodes
Mazumdar et al. [50]	Strengths	The sojourn points are selected from the base station. The mobile data collectors use a specific path for data collection.
	Limitations	The cluster head selection criteria is not determined. Cluster head and sojourn point are prone to energy exhaustion.
Akram et al. (2021) [4]	Strengths	simulation and real testbed environment evaluation performance.
	Limitations	The nodes in the network have to deal with a lot of calculations
Papi et al. (2022) [60]	Strengths	when the number of holes exceeds the mobile node, the ones with most priority are selected.
	Limitations	Evaluation is performed with only ten mobile nodes for a range of 50 to 500 static nodes.
Wu (2022) [90]	Strengths	the recovery technique uses a simple procedure and formula.
	Limitations	only one experiment was performed.

Table B.3: Strengths and Limitations of the Mobile Nodes Algorithms